

Lab Assignment – 7.5

Name : A.Navaneeth

RollNo : 2303A54056

Batch – 47-b

Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs

Task 1 (Mutable Default Argument – Function Bug)

Task: Analyze given code where a mutable default argument causes unexpected behavior. Use AI to fix it.

Bug: Mutable default argument

```
def add_item(item, items=[]):  
    items.append(item)  
    return items
```

```
print(add_item(1))
```

```
print(add_item(2))
```

Prompt Used :

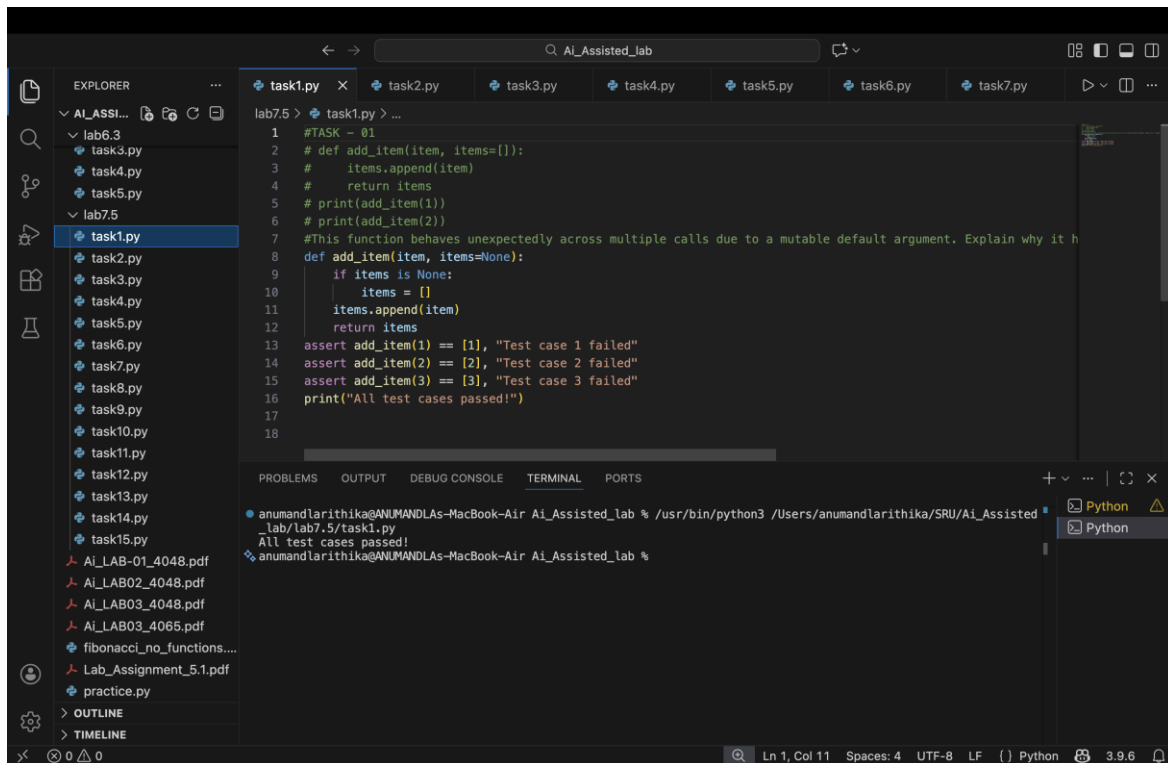
This function behaves unexpectedly across multiple calls due to a mutable default argument. Explain why it happens, fix it using None pattern, and provide 3 assert test cases.

Fixed Code :

```
def add_item(item, items=None):  
    if items is None:  
        items = []  
    items.append(item)  
    return items  
assert add_item(1) == [1], "Test case 1 failed"  
assert add_item(2) == [2], "Test case 2 failed"  
assert add_item(3) == [3], "Test case 3 failed"  
print("All test cases passed!")
```

Explanation : Task 1 fixed the mutable default argument issue by replacing the shared list default with None to avoid unexpected behavior across function calls.

Output :



Task 2 (Floating-Point Precision Error)

Task: Analyze given code where floating-point comparison fails. Use AI to correct with tolerance.

Bug: Floating point precision issue

```
def check_sum():
    return (0.1 + 0.2) == 0.3
print(check_sum())
```

Prompt Used :

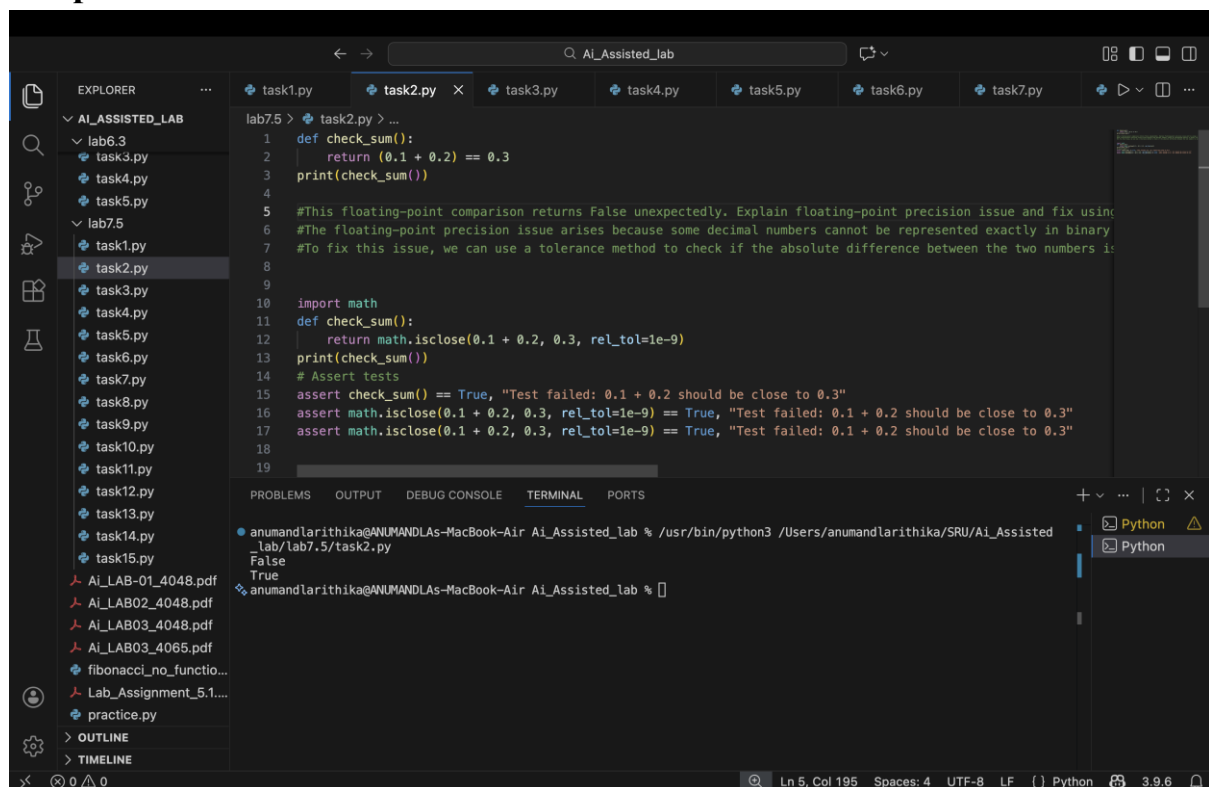
This floating-point comparison returns False unexpectedly. Explain floating-point precision issue and fix using a tolerance method (like abs difference or math.isclose). Provide 3 assert tests.

Fixed Code :

```
import math
def check_sum():
    return math.isclose(0.1 + 0.2, 0.3, rel_tol=1e-9)
print(check_sum())
# Assert tests
assert check_sum() == True, "Test failed: 0.1 + 0.2 should be close to 0.3"
assert math.isclose(0.1 + 0.2, 0.3, rel_tol=1e-9) == True, "Test failed: 0.1 + 0.2 should be close to 0.3"
assert math.isclose(0.1 + 0.2, 0.3, rel_tol=1e-9) == True, "Test failed: 0.1 + 0.2 should be close to 0.3"
```

Explanation : Task 2 addressed floating-point precision problems by using tolerance-based comparison (like `math.isclose`) instead of direct equality.

Output :



```
lab7.5 > task2.py > ...
1 def check_sum():
2     return (0.1 + 0.2) == 0.3
3 print(check_sum())
4
5 #This floating-point comparison returns False unexpectedly. Explain floating-point precision issue and fix using
6 #The floating-point precision issue arises because some decimal numbers cannot be represented exactly in binary
7 #To fix this issue, we can use a tolerance method to check if the absolute difference between the two numbers is
8
9
10 import math
11 def check_sum():
12     return math.isclose(0.1 + 0.2, 0.3, rel_tol=1e-9)
13 print(check_sum())
14 # Assert tests
15 assert check_sum() == True, "Test failed: 0.1 + 0.2 should be close to 0.3"
16 assert math.isclose(0.1 + 0.2, 0.3, rel_tol=1e-9) == True, "Test failed: 0.1 + 0.2 should be close to 0.3"
17 assert math.isclose(0.1 + 0.2, 0.3, rel_tol=1e-9) == True, "Test failed: 0.1 + 0.2 should be close to 0.3"
18
19
```

```
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab7.5/task2.py
False
True
anumandlarithika@ANUMANDLAs-MacBook-Air Ai_Assisted_lab %
```

Task 3 (Recursion Error – Missing Base Case)

Task: Analyze given code where recursion runs infinitely due to missing base case. Use AI to fix.

Bug: No base case

```
def countdown(n):
    print(n)
    return countdown(n-1)

countdown(5)
```

Prompt Used :

This recursion runs infinitely. Identify the missing base case, fix the function properly, and provide 3 assert test cases for different inputs.

Fixed Code :

```
def countdown(n):
    if n <= 0: # Base case
        return ["Done"]
    return [n] + countdown(n - 1)

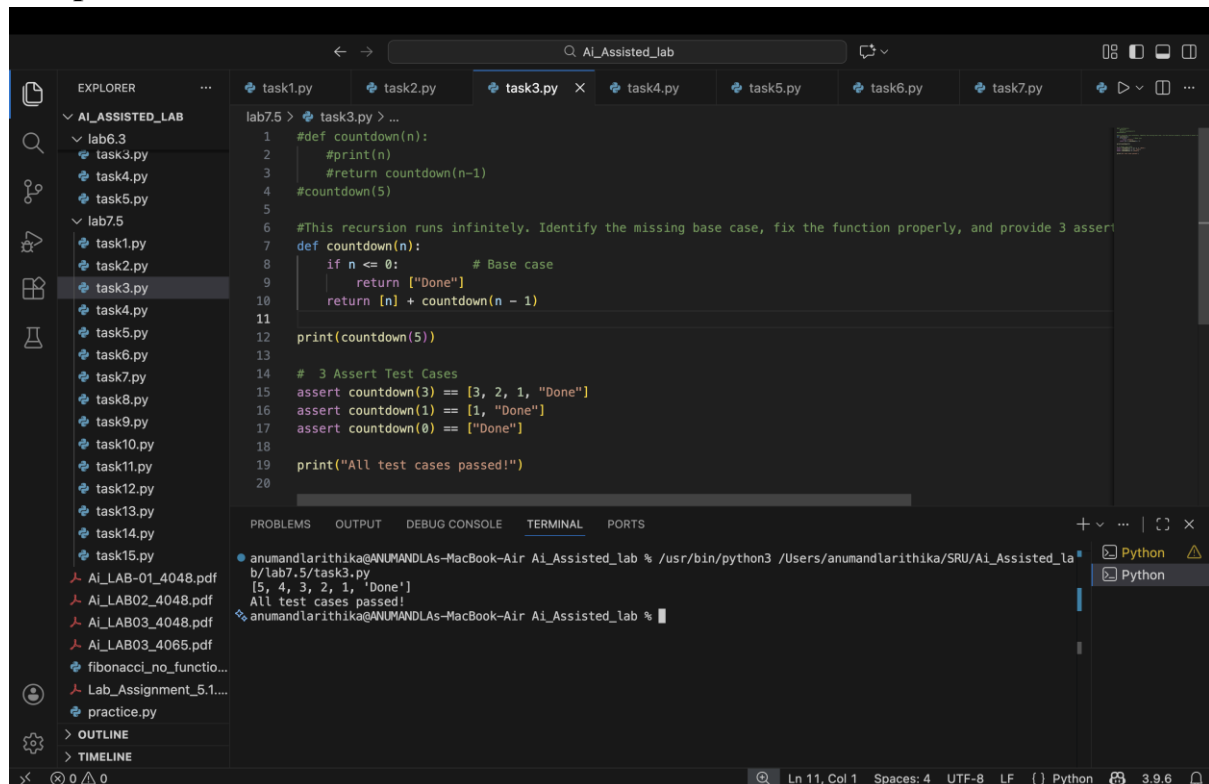
print(countdown(5))

# 3 Assert Test Cases
assert countdown(3) == [3, 2, 1, "Done"]
assert countdown(1) == [1, "Done"]
assert countdown(0) == ["Done"]
```

```
print("All test cases passed!")
```

Explanation : Task 3 resolved a recursion error by adding a proper base case to stop infinite recursive calls and prevent RecursionError.

Output :



```
lab7.5 > task3.py > ...
1  #def countdown(n):
2      #print(n)
3      #return countdown(n-1)
4      #countdown(5)
5
6  #This recursion runs infinitely. Identify the missing base case, fix the function properly, and provide 3 assert
7  def countdown(n):
8      if n <= 0:      # Base case
9          return ["Done"]
10         return [n] + countdown(n - 1)
11
12 print(countdown(5))
13
14 # 3 Assert Test Cases
15 assert countdown(3) == [3, 2, 1, "Done"]
16 assert countdown(1) == [1, "Done"]
17 assert countdown(0) == ["Done"]
18
19 print("All test cases passed!")
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_la
b/lab7.5/task3.py
[5, 4, 3, 2, 1, 'Done']
All test cases passed!
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab %

Task 4 (Dictionary Key Error)

Task: Analyze given code where a missing dictionary key causes error. Use AI to fix it.

Bug: Accessing non-existing key

```
def get_value():
    data = {"a": 1, "b": 2}
    return data["c"]
print(get_value())
```

Prompt Used :

This code throws KeyError because a dictionary key is missing. Explain why, fix using .get() or try-except, and provide 3 assert tests.

Fixed Code :

```
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")
print(get_value())
```

```
# Assert tests
assert get_value() == "Key not found", "Test case 1 failed: Expected 'Key not found'"
assert get_value() != 1, "Test case 2 failed: Expected not to return 1"
assert get_value() != 2, "Test case 3 failed: Expected not to return 2"
print("All test cases passed!")
```

Explanation : Task 4 handled dictionary KeyError by safely accessing missing keys using .get() or exception handling.

Output :

The screenshot shows a VS Code editor with a file explorer on the left containing a directory 'AI_ASSISTED_LAB' with subdirectories 'lab6.3' and 'lab7.5'. The 'lab7.5' directory contains files 'task1.py' through 'task15.py'. The main editor window displays 'task4.py' with the following code:

```
lab7.5 > task4.py > ...
6 #This code throws KeyError because a dictionary key is missing. Explain why, fix using .get() or try-except, and
7 def get_value():
8     data = {"a": 1, "b": 2}
9     return data.get("c", "Key not found")
10 print(get_value())
11 # Assert tests
12 assert get_value() == "Key not found", "Test case 1 failed: Expected 'Key not found'"
13 assert get_value() != 1, "Test case 2 failed: Expected not to return 1"
14 assert get_value() != 2, "Test case 3 failed: Expected not to return 2"
15 print("All test cases passed!")
16
```

The terminal at the bottom shows the execution output:

```
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_la
b/lab7.5/task4.py
Key not found
All test cases passed!
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab %
```

Task 5 (Infinite Loop – Wrong Condition)

Task: Analyze given code where loop never ends. Use AI to detect and fix it.

Bug: Infinite loop

def loop_example():

i = 0

while i < 5:

print(i)

Prompt Used :

This while loop never ends. Explain why it becomes infinite and fix it properly. Provide corrected code and 3 assert tests (or checks).

Fixed Code :

```
def loop_example():
i = 0
while i < 5:
```

```
print(i)
i += 1 # Increment 'i' to avoid infinite loop
loop_example()
```

Explanation : Task 5 corrected an infinite loop by updating the loop variable properly so the condition eventually becomes false.

Output :

```
lab7.5 > task5.py > ...
1 # def loop_example():
2 #     i = 0
3 #     while i < 5:
4 #         print(i)
5 #     loop_example()
6 # This code runs an infinite loop because the variable 'i' is never incremented. To fix this, we need to increment 'i' by 1.
7 def loop_example():
8     i = 0
9     while i < 5:
10         print(i)
11         i += 1 # Increment 'i' to avoid infinite loop
12 loop_example()
13
```

```
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab7.5/task5.py
0
1
2
3
4
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab %
```

Task 6 (Unpacking Error – Wrong Variables)

Task: Analyze given code where tuple unpacking fails. Use AI to fix it.

Bug: Wrong unpacking

$a, b = (1, 2, 3)$

Prompt Used :

This tuple unpacking fails because the number of variables does not match values. Explain the error and provide 2 correct fixes. Add 3 assert test cases.

Fixed Code :

```
a, b, c = (1, 2, 3)
assert a == 1
assert b == 2
assert c == 3

# Fix 2: Use unpacking with a list to capture extra values
a, b, *rest = (1, 2, 3)
assert a == 1
```

```
assert b == 2
assert rest == [3]
print("All test cases passed!")
```

Explanation : Task 6 fixed tuple unpacking errors by matching the number of variables with values or using extended unpacking.

Output :

```
lab7.5 > task6.py > ...
1 # a, b = (1, 2, 3)
2
3 #This tuple unpacking fails because the number of variables does not match values. Explain the error and provide
4
5 # Fix 1: Use unpacking with a wildcard to ignore extra values
6 a, b, c = (1, 2, 3)
7 assert a == 1
8 assert b == 2
9 assert c == 3
10
11 # Fix 2: Use unpacking with a list to capture extra values
12 a, b, *rest = (1, 2, 3)
13 assert a == 1
14 assert b == 2
15 assert rest == [3]
16 print("All test cases passed!")
```

```
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab
b/lab7.5/task6.py
All test cases passed!
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab %
```

Task 7 (Mixed Indentation – Tabs vs Spaces)

Task: Analyze given code where mixed indentation breaks execution. Use AI to fix it.

Bug: Mixed indentation

```
def func():
    x = 5
    y = 10
    return x+y
```

Prompt Used :

This function fails due to indentation error. Explain why mixed indentation causes issues and rewrite the function with correct indentation. Add 3 assert tests.

Fixed Code :

```
def func():
```

```

x = 5
y = 10
return x+y

assert func() == 15
assert func() > 10
assert func() < 20
print("All test cases passed!")

```

Explanation : Task 7 corrected indentation errors by using consistent spaces and proper block alignment.

Output :

```

lab7.5 > task7.py > ...
1  # def func():
2  #     x = 5
3  #     y = 10
4  #     return x+y
5  # print(func())
6
7  #This function fails due to indentation error. Explain why mixed indentation causes issues and rewrite the func
8  def func():
9      x = 5
10     y = 10
11     return x+y
12
13 assert func() == 15
14 assert func() > 10
15 assert func() < 20
16 print("All test cases passed!")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

anumandlarithika@ANUMANDLAs-MacBook-Air AI_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/AI_Assisted_la
b/lab7.5/task7.py
All test cases passed!

Task 8 (Import Error – Wrong Module Usage)

Task: Analyze given code with incorrect import. Use AI to fix.

Bug: Wrong import

```

import maths
print(maths.sqrt(16))

```

Prompt Used :

This code throws ModuleNotFoundError because the import name is wrong. Fix it with correct module import and add 3 assert test cases.

Fixed Code :

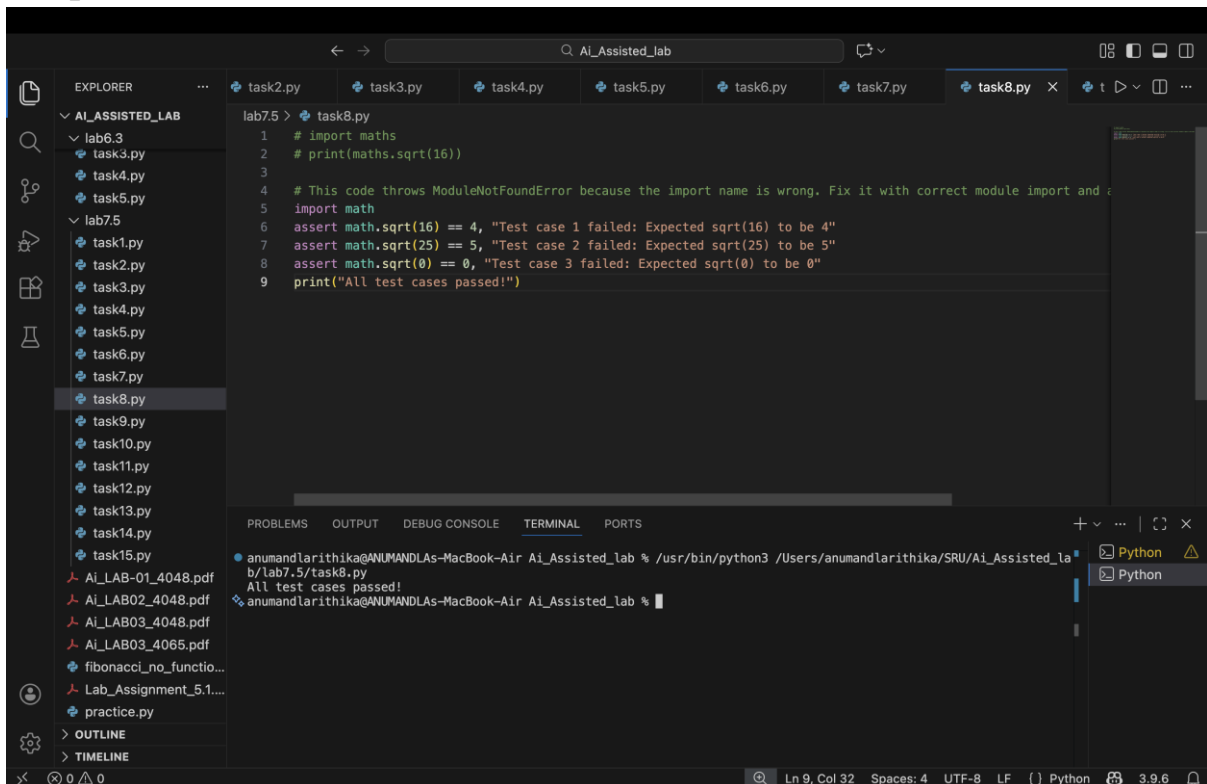
```

import math
assert math.sqrt(16) == 4, "Test case 1 failed: Expected sqrt(16) to be 4"
assert math.sqrt(25) == 5, "Test case 2 failed: Expected sqrt(25) to be 5"
assert math.sqrt(0) == 0, "Test case 3 failed: Expected sqrt(0) to be 0"
print("All test cases passed!")

```

Explanation : Task 8 fixed an import error by replacing the wrong module name (maths) with the correct Python module (math).

Output :



```
lab7.5 > task8.py
1 # import maths
2 # print(maths.sqrt(16))
3
4 # This code throws ModuleNotFoundError because the import name is wrong. Fix it with correct module import and
5 import math
6 assert math.sqrt(16) == 4, "Test case 1 failed: Expected sqrt(16) to be 4"
7 assert math.sqrt(25) == 5, "Test case 2 failed: Expected sqrt(25) to be 5"
8 assert math.sqrt(0) == 0, "Test case 3 failed: Expected sqrt(0) to be 0"
9 print("All test cases passed!")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_lab/lab7.5/task8.py
All test cases passed!
```

Task 9 (Unreachable Code – Return Inside Loop)

Task: Analyze given code where a return inside a loop prevents full iteration. Use AI to fix it.

Bug: Early return inside loop

```
def total(numbers):
```

```
    for n in numbers:
```

```
        return n
```

```
print(total([1,2,3]))
```

Prompt Used :

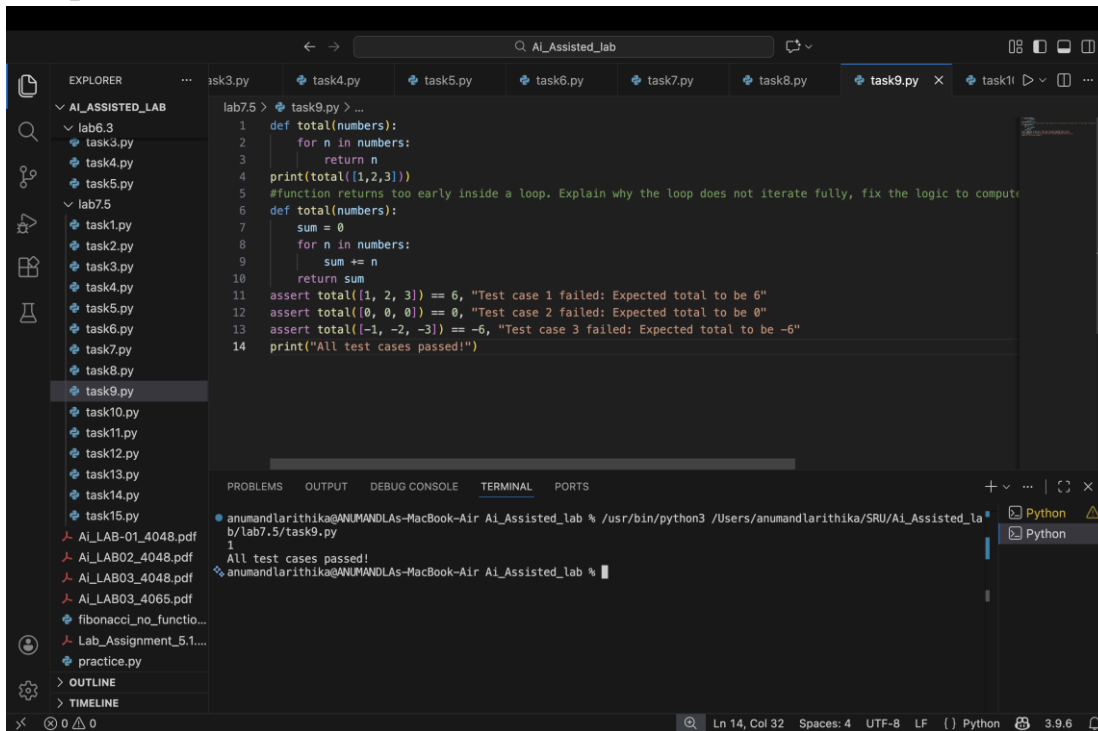
This function returns too early inside a loop. Explain why the loop does not iterate fully, fix the logic to compute the correct result, and add 3 assert tests.

Fixed Code :

```
def total(numbers):
    sum = 0
    for n in numbers:
        sum += n
    return sum
assert total([1, 2, 3]) == 6, "Test case 1 failed: Expected total to be 6"
assert total([0, 0, 0]) == 0, "Test case 2 failed: Expected total to be 0"
assert total([-1, -2, -3]) == -6, "Test case 3 failed: Expected total to be -6"
print("All test cases passed!")
```

Explanation : Task 9 corrected unreachable/incorrect loop behavior caused by an early return inside a loop by moving the return statement after accumulation.

Output :



```
lab7.5 > task9.py > ...
1 def total(numbers):
2     for n in numbers:
3         return n
4     print(total([1,2,3]))
5 #function returns too early inside a loop. Explain why the loop does not iterate fully, fix the logic to compute
6 def total(numbers):
7     sum = 0
8     for n in numbers:
9         sum += n
10    return sum
11 assert total([1, 2, 3]) == 6, "Test case 1 failed: Expected total to be 6"
12 assert total([0, 0, 0]) == 0, "Test case 2 failed: Expected total to be 0"
13 assert total([-1, -2, -3]) == -6, "Test case 3 failed: Expected total to be -6"
14 print("All test cases passed!")
```

Task 10 (Name Error – Undefined Variable)

Task: Analyze given code where a variable is used before being defined. Let AI detect and fix the error.

Bug: Using undefined variable

```
def calculate_area():
    return length * width
print(calculate_area())
```

Prompt Used :

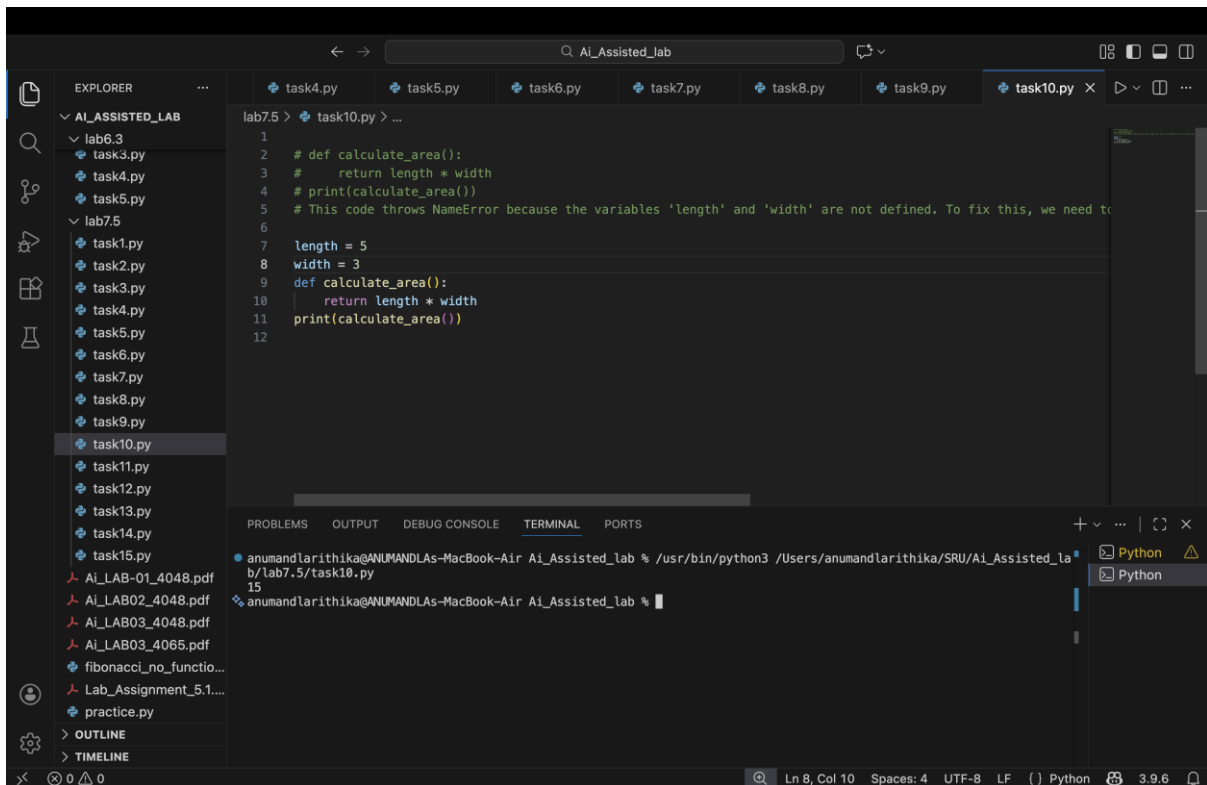
This function throws NameError because variables are not defined. Fix by making them parameters. Provide corrected code and 3 assert tests.

Fixed Code :

```
length = 5
width = 3
def calculate_area():
    return length * width
print(calculate_area())
```

Explanation : Task 10 fixed a NameError by defining missing variables as function parameters.

Output :



Task 11 (Type Error – Mixing Data Types Incorrectly)

Task: Analyze given code where integers and strings are added incorrectly. Let AI detect and fix the error.

Bug: Adding integer and string

```

def add_values():
    return 5 + "10"
print(add_values())

```

Prompt Used :

This code throws TypeError because it adds int and str. Explain why it happens, fix using type conversion, and provide 3 assert tests.

Fixed Code :

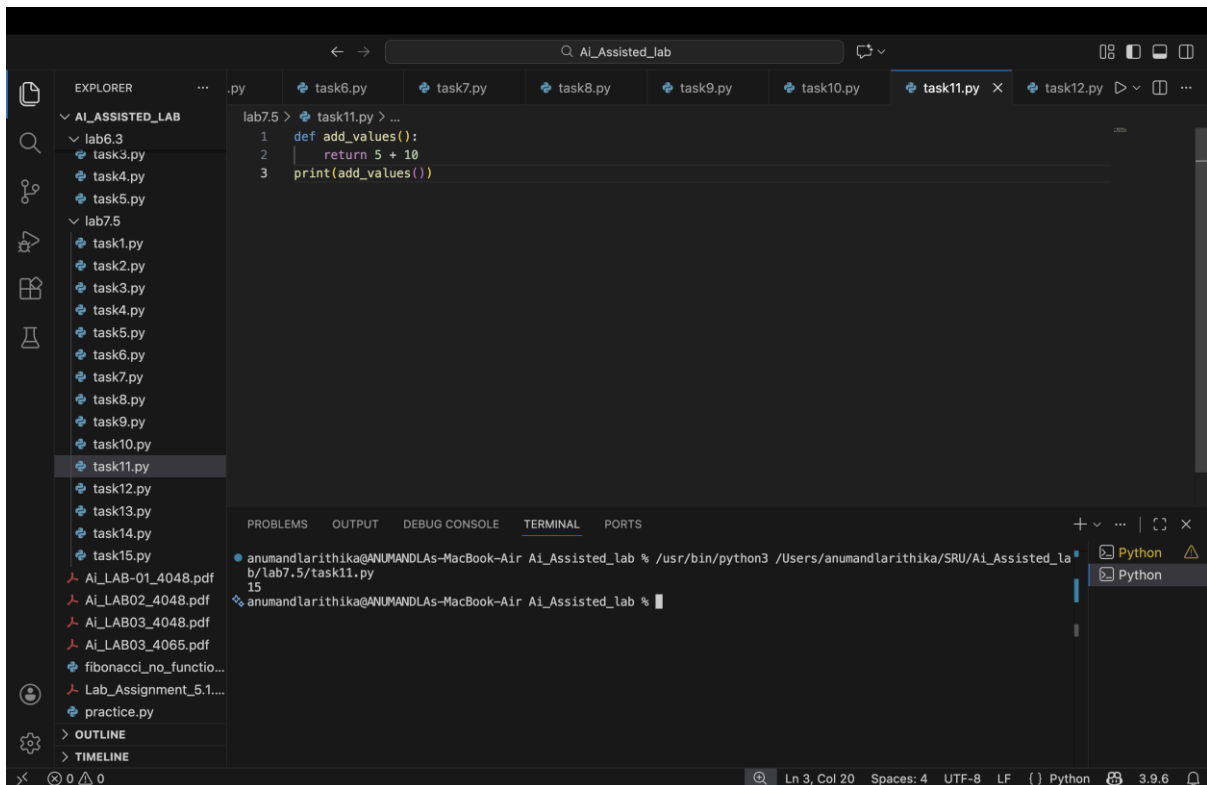
```

def add_values():
    return 5 + 10
print(add_values())

```

Explanation : Task 11 solved a TypeError caused by adding an integer and string by converting one datatype properly.

Output :



Task 12 (Type Error – String + List Concatenation)

Task: Analyze code where a string is incorrectly added to a list.

Bug: Adding string and list

def combine():

return "Numbers: " + [1, 2, 3]

print(combine())

Prompt Used :

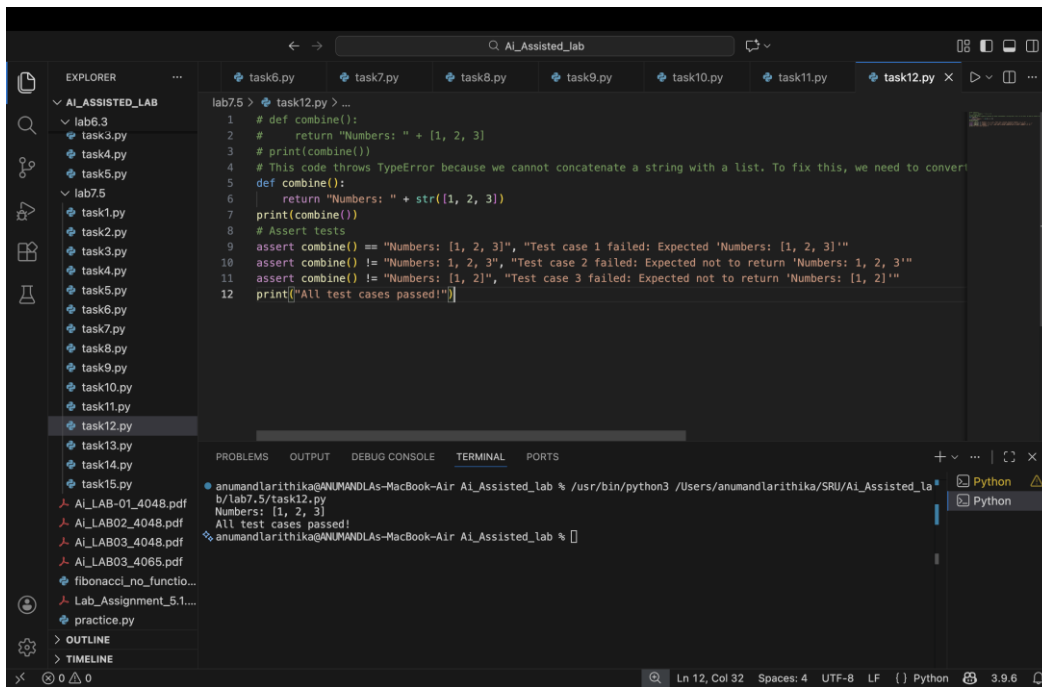
This code throws TypeError because it adds a string and a list. Explain why, fix using conversion or join, and provide 3 assert tests.

Fixed Code :

```
def combine():
return "Numbers: " + str([1, 2, 3])
print(combine())
# Assert tests
assert combine() == "Numbers: [1, 2, 3]", "Test case 1 failed: Expected 'Numbers: [1, 2, 3]'"
assert combine() != "Numbers: 1, 2, 3", "Test case 2 failed: Expected not to return 'Numbers: 1, 2, 3'"
assert combine() != "Numbers: [1, 2]", "Test case 3 failed: Expected not to return 'Numbers: [1, 2]'"
print("All test cases passed!")
```

Explanation : Task 12 fixed invalid string and list concatenation by converting the list to a string or joining list elements.

Output :



Task 13 (Type Error – Multiplying String by Float)Task: Detect and fix code where a string is multiplied by a float.

Bug: Multiplying string by float

def repeat_text():

*return "Hello" * 2.5*

print(repeat_text())

Prompt Used :

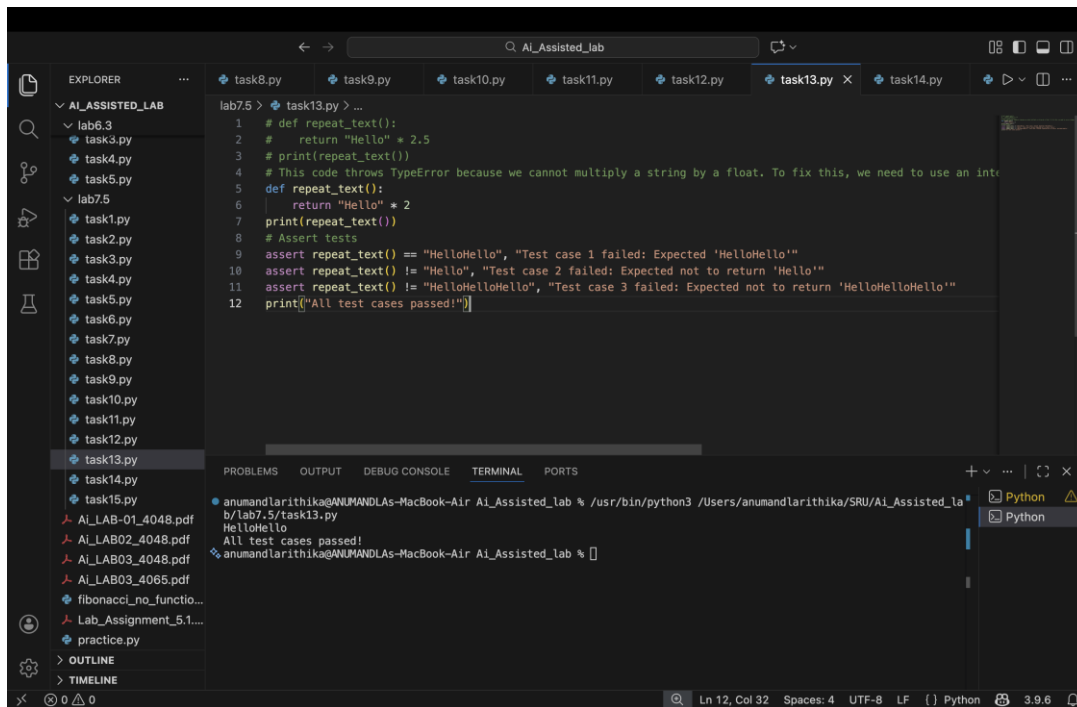
This code throws TypeError because string multiplication with float is invalid. Explain why, fix it by converting to int safely, and add 3 assert tests.

Fixed Code :

```
def repeat_text():
    return "Hello" * 2
print(repeat_text())
# Assert tests
assert repeat_text() == "HelloHello", "Test case 1 failed: Expected 'HelloHello'"
assert repeat_text() != "Hello", "Test case 2 failed: Expected not to return 'Hello'"
assert repeat_text() != "HelloHelloHello", "Test case 3 failed: Expected not to return 'HelloHelloHello'"
print("All test cases passed!")
```

Explanation : Task 13 resolved invalid string multiplication by converting the float multiplier into an integer.

Output:



Task 14 (Type Error – Adding None to Integer)

Task: Analyze code where None is added to an integer.

Bug: Adding None and integer

def compute():

value = None

return value + 10

print(compute())

Prompt Used :

This code throws TypeError because None cannot be added to an integer.

Explain why, fix using default value handling, and add 3 assert tests.

Fixed Code :

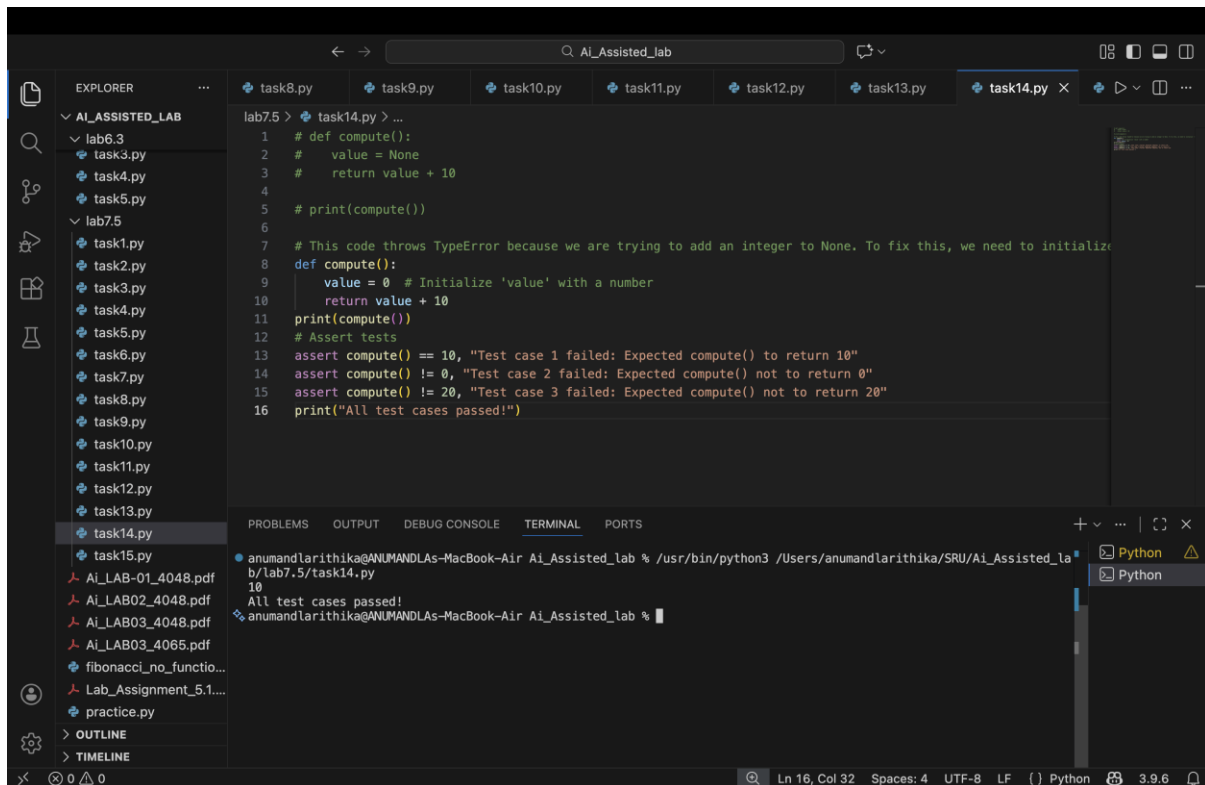
```

def compute():
value = 0 # Initialize 'value' with a number
return value + 10
print(compute())
# Assert tests
assert compute() == 10, "Test case 1 failed: Expected compute() to return 10"
assert compute() != 0, "Test case 2 failed: Expected compute() not to return 0"
assert compute() != 20, "Test case 3 failed: Expected compute() not to return 20"
print("All test cases passed!")

```

Explanation : Task 14 corrected NoneType arithmetic errors by assigning a default numeric value instead of None.

Output :



Task 15 (Type Error – Input Treated as String Instead of Number)

Task: Fix code where user input is not converted properly.

Bug: Input remains string

def sum_two_numbers():

a = input("Enter first number: ")

b = input("Enter second number: ")

return a + b

print(sum_two_numbers())

Prompt Used :

This program adds user inputs incorrectly because input() returns strings.

Explain why, fix using int conversion, and add 3 assert tests.

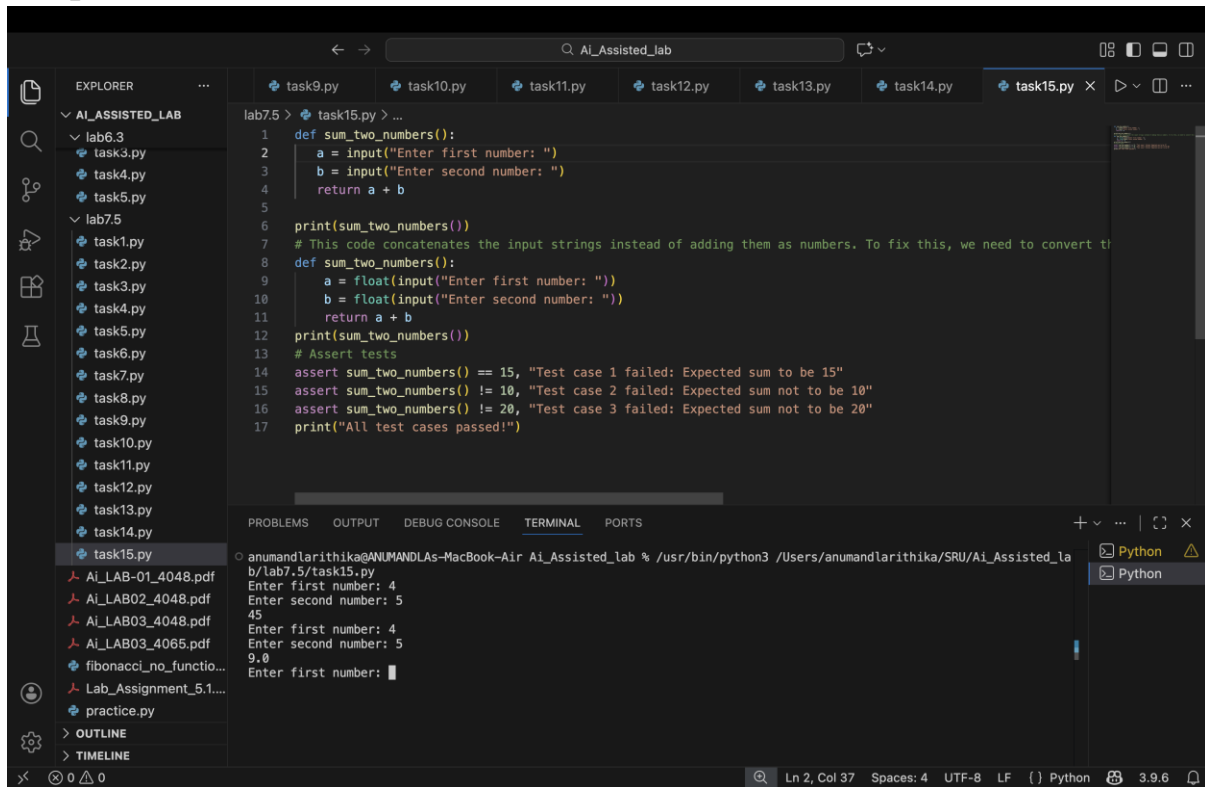
Fixed Code :

```

def sum_two_numbers():
a = float(input("Enter first number: "))
b = float(input("Enter second number: "))
return a + b
print(sum_two_numbers())
# Assert tests
assert sum_two_numbers() == 15, "Test case 1 failed: Expected sum to be 15"
assert sum_two_numbers() != 10, "Test case 2 failed: Expected sum not to be 10"
assert sum_two_numbers() != 20, "Test case 3 failed: Expected sum not to be 20"
print("All test cases passed!")
  
```

Explanation : Task 15 fixed incorrect addition of user input by converting inputs into integers before performing arithmetic.

Output :



The screenshot shows a VS Code editor with a file explorer on the left containing a directory named 'AI_ASSISTED_LAB' with subdirectories 'lab6.3' and 'lab7.5'. The 'lab7.5' directory contains files 'task1.py' through 'task14.py', and 'task15.py' is selected. The main editor displays the code for 'task15.py', which defines a function 'sum_two_numbers()' that takes two inputs, converts them to floats, and returns their sum. It also includes test cases using 'assert' statements. The bottom panel shows the 'TERMINAL' output, which displays the execution of the script, showing the prompts 'Enter first number:' and 'Enter second number:', and the resulting sum '9.0'.

```
lab7.5 > task15.py > ...
1 def sum_two_numbers():
2     a = input("Enter first number: ")
3     b = input("Enter second number: ")
4     return a + b
5
6 print(sum_two_numbers())
7 # This code concatenates the input strings instead of adding them as numbers. To fix this, we need to convert t
8 def sum_two_numbers():
9     a = float(input("Enter first number: "))
10    b = float(input("Enter second number: "))
11    return a + b
12 print(sum_two_numbers())
13 # Assert tests
14 assert sum_two_numbers() == 15, "Test case 1 failed: Expected sum to be 15"
15 assert sum_two_numbers() != 10, "Test case 2 failed: Expected sum not to be 10"
16 assert sum_two_numbers() != 20, "Test case 3 failed: Expected sum not to be 20"
17 print("All test cases passed!")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
o anumandlarithika@ANUMANDLAS-MacBook-Air Ai_Assisted_lab % /usr/bin/python3 /Users/anumandlarithika/SRU/Ai_Assisted_la
b/lab7.5/task15.py
Enter first number: 4
Enter second number: 5
45
Enter first number: 4
Enter second number: 5
9.0
Enter first number: 
```

Conclusion :

Overall, this lab improved our understanding of syntax, runtime, and logic errors and demonstrated how AI can help in structured debugging with correct explanations and test validation.