AI Assisted Coding

Assignment 7.5

Name: V.Harivamsh

Hall ticket no: 2303A51266

Batch no: 19

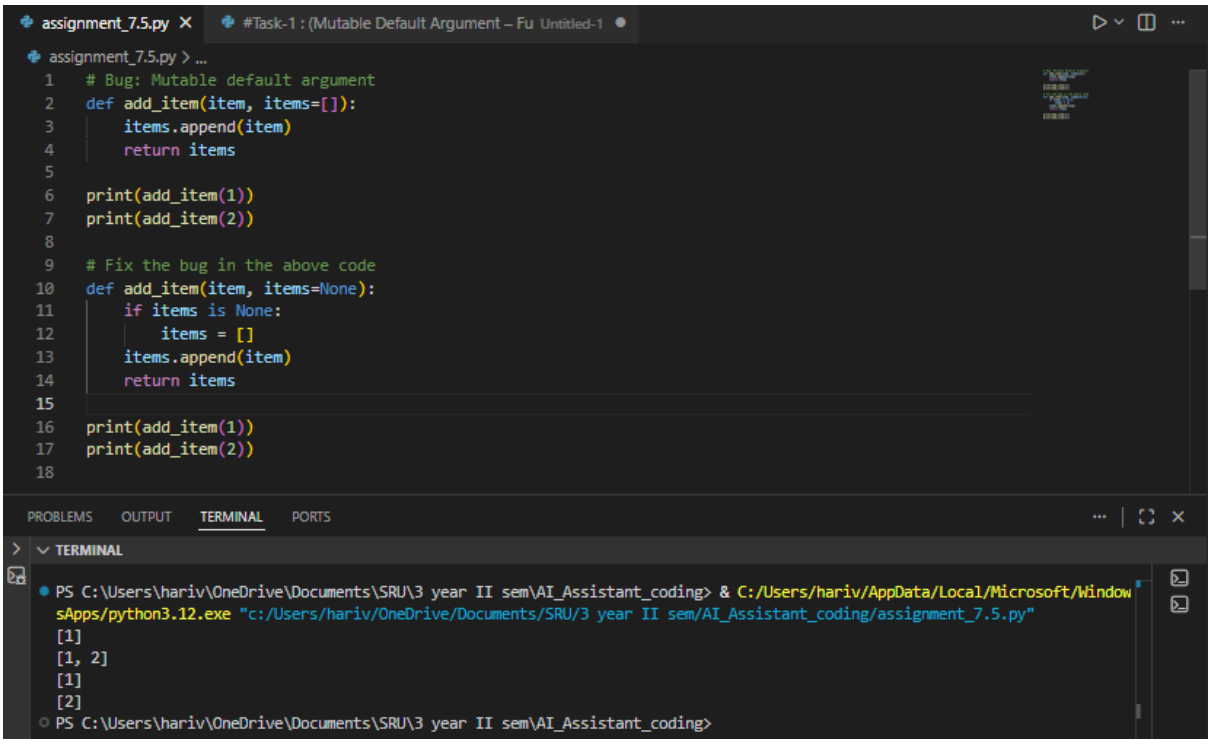
Task 1:

Prompt:

#Bug: Mutable default argument

#Fix the bug in the above code

Code & Output:



Explanation:

The AI correctly recognized that using a mutable object like a list as a default argument causes the same list to be shared across multiple function calls, leading to unexpected value accumulation. In the corrected version, the AI initializes the list inside the function when no argument is passed, ensuring a new list is created each time. This method avoids unwanted side effects and follows recommended best practices for designing Python functions.

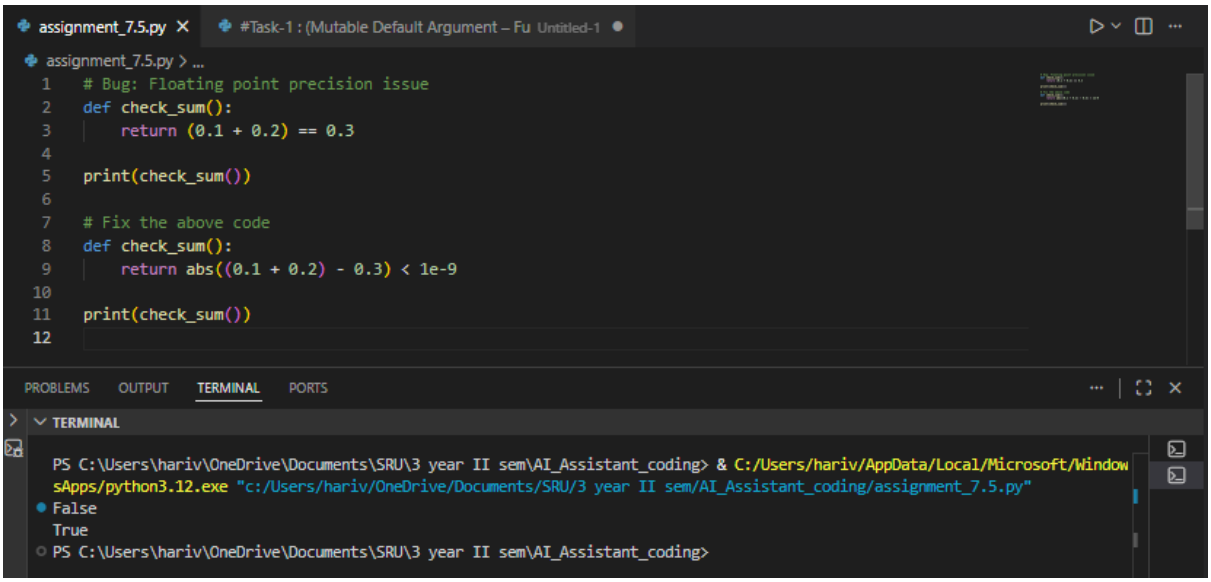
Task 2: Floating-Point Precision Error

Prompt:

#Bug: Floating point precision issue

#Fix the above code

Code & Output:



Explanation:

The AI recognized that floating-point values cannot always be compared directly because of precision issues in their binary representation. Rather than relying on exact equality, it recommended comparing the difference between the values within a small acceptable range. This approach makes numerical calculations more accurate and reliable.

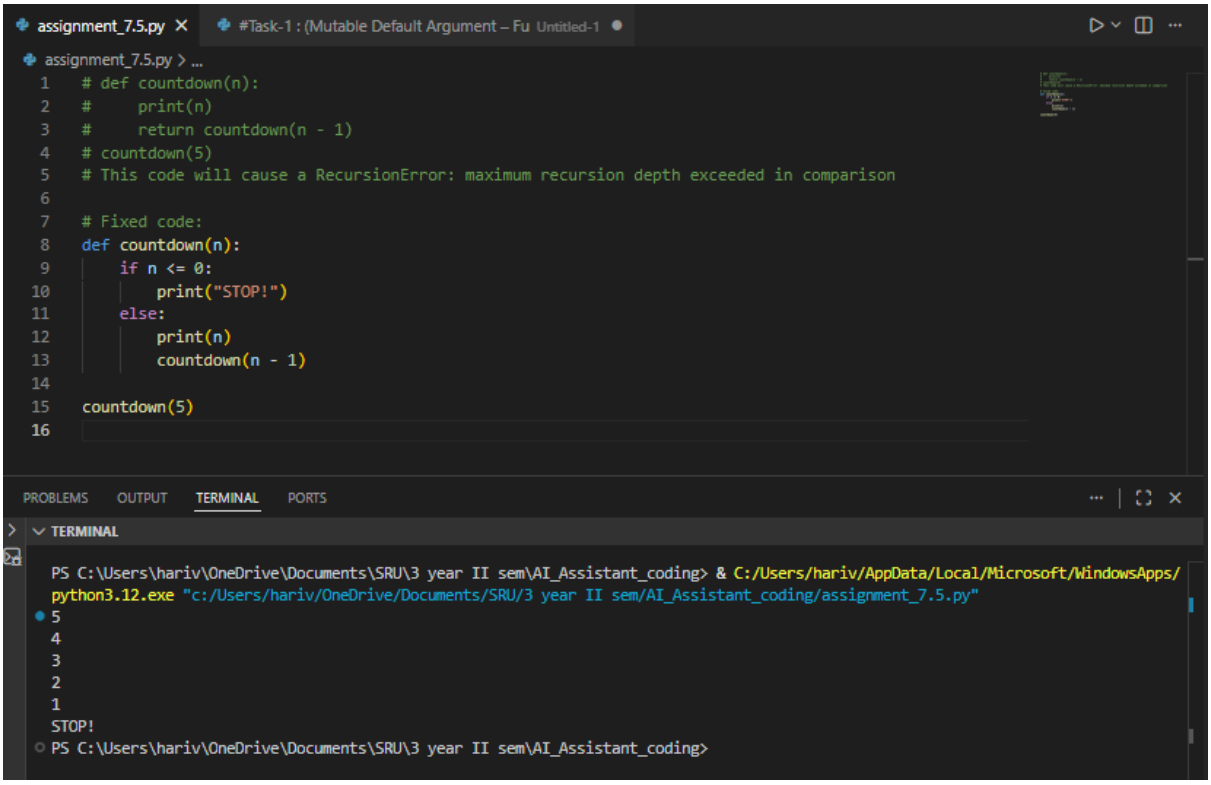
Task 3: Recursion Error – Missing Base Case

Prompt:

#This code will cause a RecursionError: maximum recursion depth exceeded in comparison

#Fixed Code:

Code & Output:



Explanation:

The AI noticed that the recursive function was missing a stopping condition, which led to infinite recursion and eventually a stack overflow. By introducing a proper base case, the AI made sure the recursion ends at the right point. This correction highlights how essential exit conditions are when designing recursive algorithms.

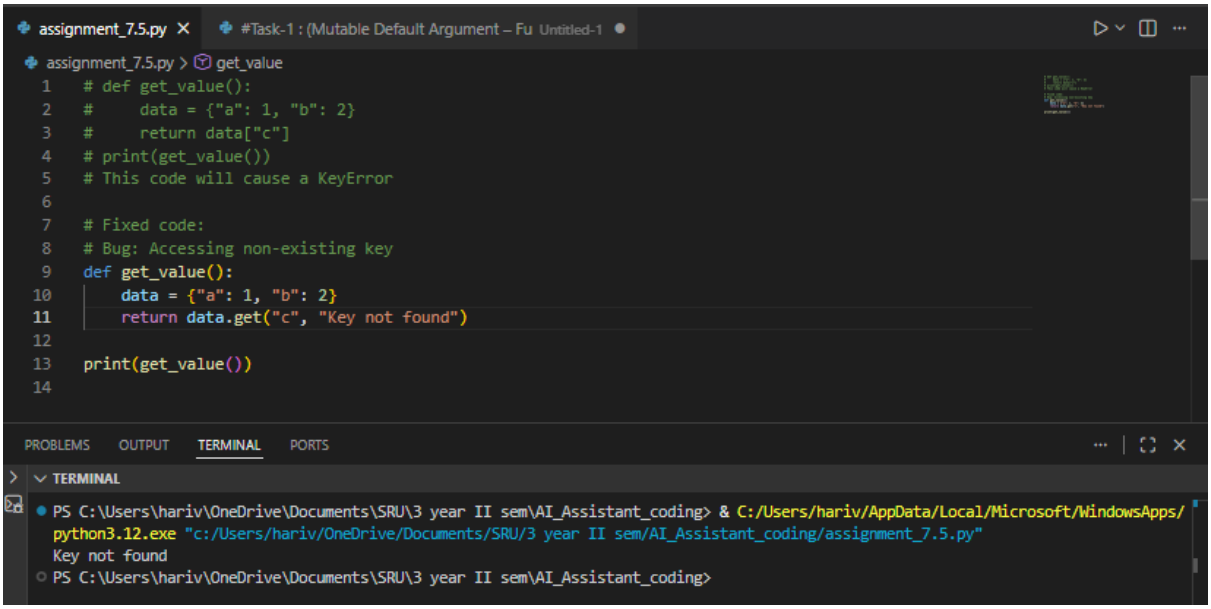
Task 4: Dictionary Key Error

Prompt:

#Fixed Code:

#Bug: Accessing non-existing key

Code & Output:



Explanation:

The AI understood that trying to access a dictionary key that does not exist results in a **KeyError**. It recommended safer approaches like using the .get() method or properly handling the exception. These alternatives make the program more robust and help avoid unexpected crashes during execution.

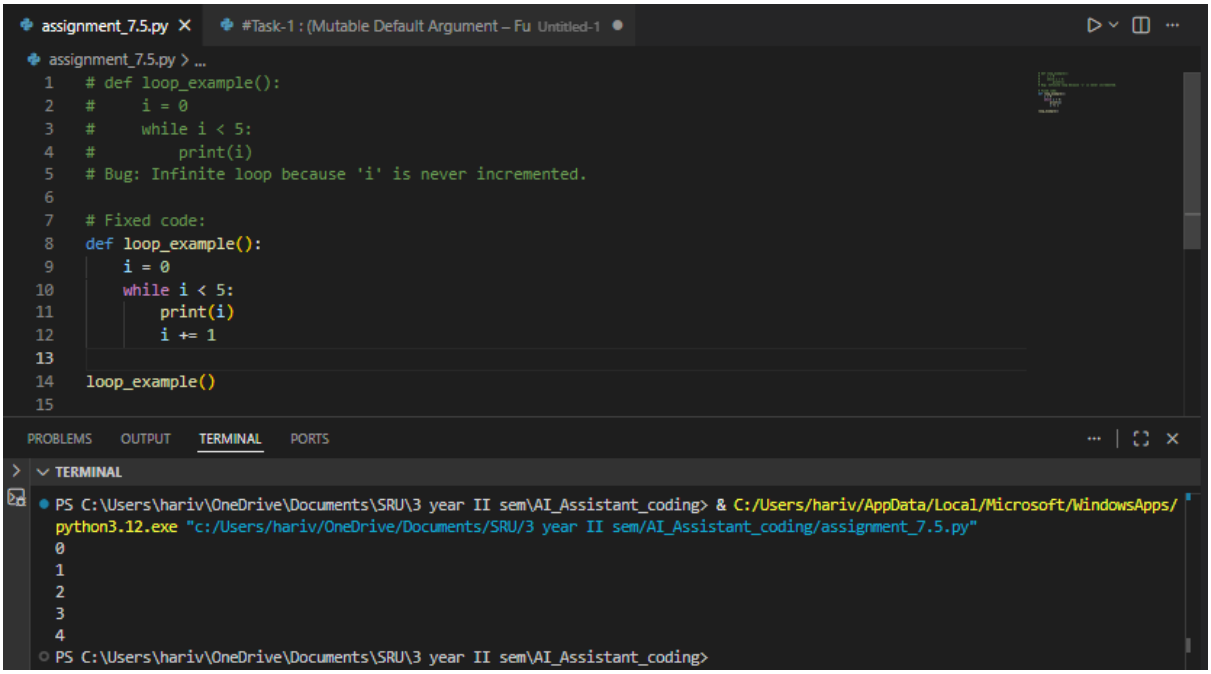
Task 5: Infinite Loop – Wrong Condition

Prompt:

#Bug: Infinite loop because ‘i’ is never incremented.

#Fixed Code:

Code & Output:



Explanation:

The AI detected that the loop variable was not being updated, which resulted in an infinite loop. By adding the appropriate increment step, the loop now moves steadily toward its termination condition. This correction emphasizes how crucial it is to update loop control variables properly.

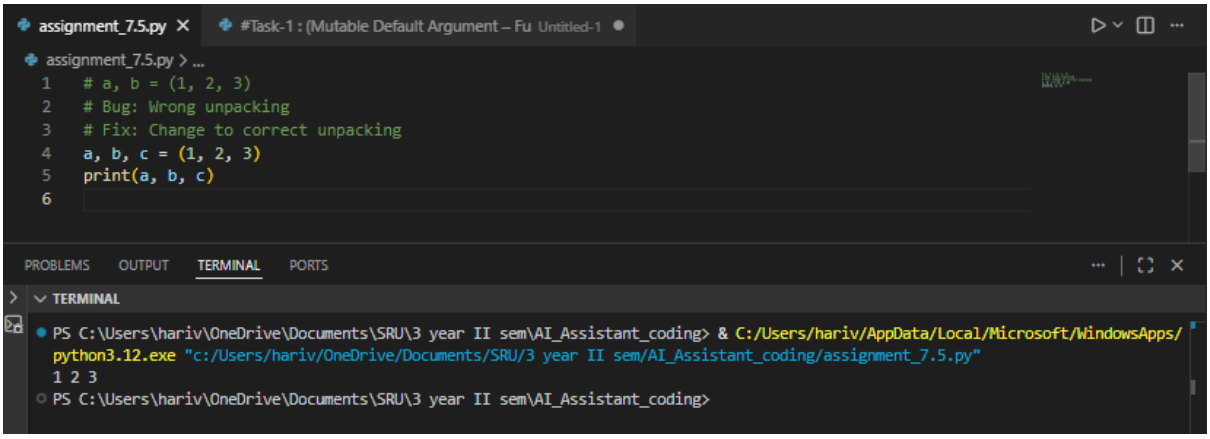
Task 6: Unpacking Error – Wrong Variables

Prompt:

#Bug: Wrong unpacking

#Fix : Change to correct unpacking

Code & Output:



Explanation:

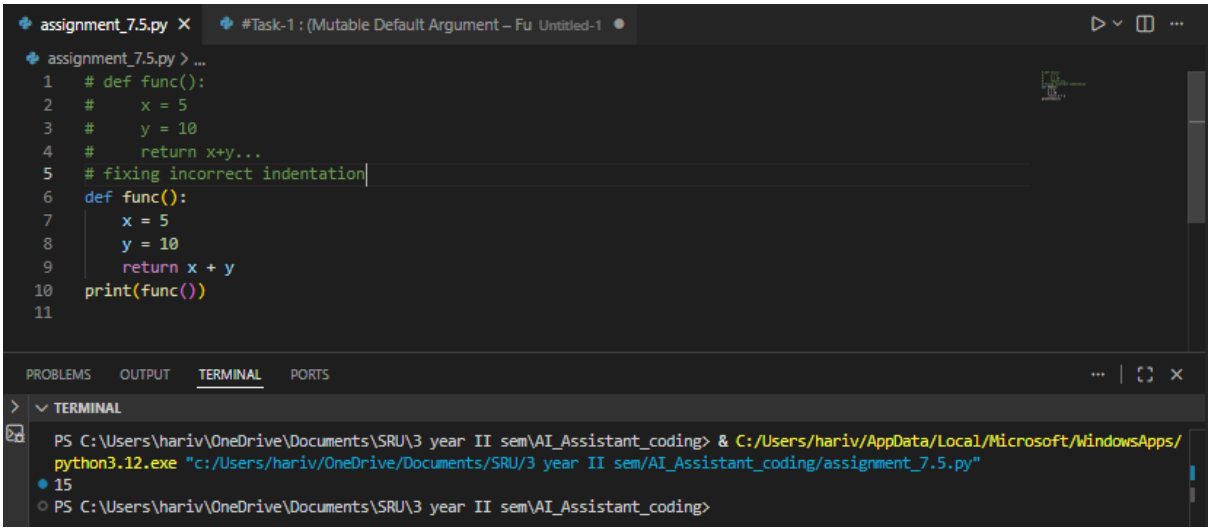
The AI identified that there was a mismatch between the number of variables and the values being unpacked from the tuple. It suggested fixing this either by adding more variables or by using a placeholder variable (_) for values that are not needed. This approach allows the tuple to be unpacked correctly and avoids runtime errors.

Task 7: Mixed Indentation – Tabs vs Spaces

Prompt:

#Fixing incorrect indentation

Code & Output:



Explanation:

The AI identified inconsistent indentation as the main reason for the error. Since Python depends heavily on indentation to define code blocks, the issue caused the program to fail. By fixing the spacing and making the indentation consistent throughout the function, the AI ensured proper execution and also improved the overall readability of the code.

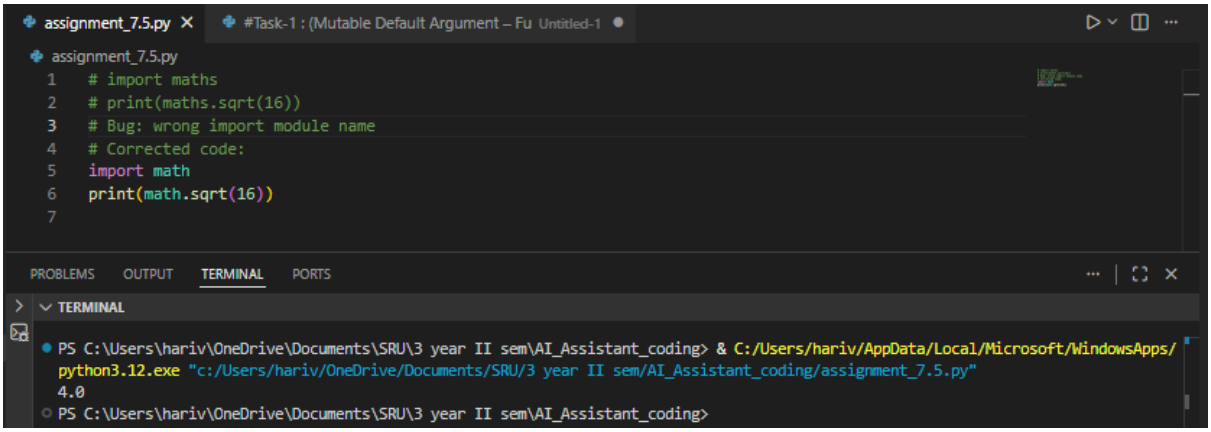
Task 8: Import Error – Wrong Module Usage

Prompt:

#Bug: Wrong import module name

#Corrected Code:

Code & Output:



Explanation:

The AI realized that the module name being imported was incorrect. It recommended using the standard Python **math** module instead. This correction fixes the import error and enables the program to access mathematical functions properly.