

AI Assisted Coding

Assignment 7.5

Name: V.Harivamsh

Hall ticket no: 2303A51266

Batch no: 19

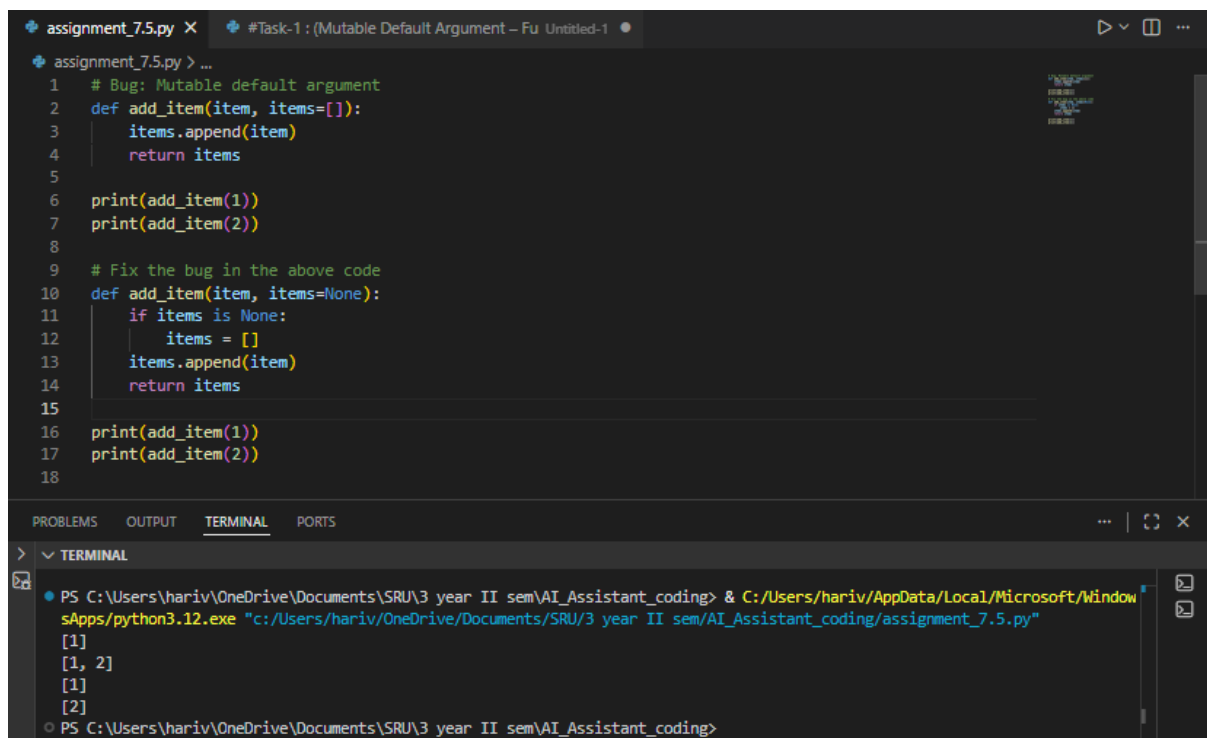
Task 1:

Prompt:

#Bug: Mutable default argument

#Fix the bug in the above code

Code & Output:



```
assignment_7.5.py X #Task-1 : (Mutable Default Argument – Fu Untitled-1
assignment_7.5.py > ...
1 # Bug: Mutable default argument
2 def add_item(item, items=[]):
3     items.append(item)
4     return items
5
6 print(add_item(1))
7 print(add_item(2))
8
9 # Fix the bug in the above code
10 def add_item(item, items=None):
11     if items is None:
12         items = []
13     items.append(item)
14     return items
15
16 print(add_item(1))
17 print(add_item(2))
18

PROBLEMS OUTPUT TERMINAL PORTS
> TERMINAL
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"
[1]
[1, 2]
[1]
[2]
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

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.

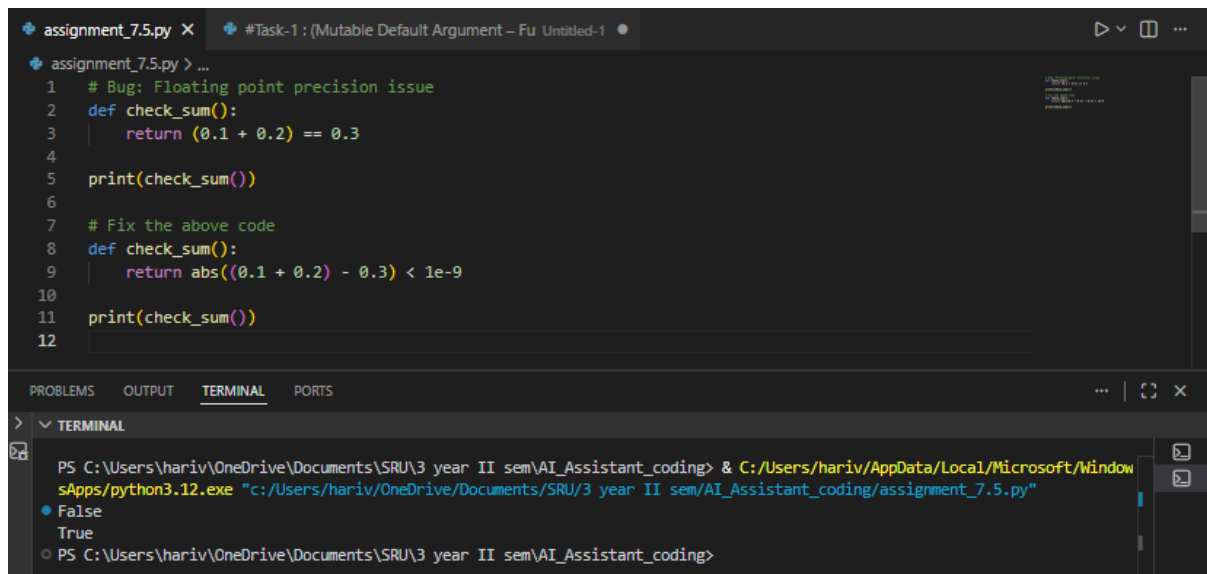
Task 2: Floating-Point Precision Error

Prompt:

#Bug: Floating point precision issue

#Fix the above code

Code & Output:



The screenshot shows a code editor with two tabs: 'assignment_7.5.py' and '#Task-1: (Mutable Default Argument - Fu Untitled-1)'. The 'assignment_7.5.py' tab is active, displaying a Python script. The script has two versions of a function 'check_sum()'. The first version (lines 2-5) returns 'True' because '0.1 + 0.2' is not exactly equal to '0.3'. The second version (lines 8-11) returns 'False' because the absolute difference between '0.1 + 0.2' and '0.3' is less than '1e-9'. The terminal at the bottom shows the command to run the script, and the output is 'False' followed by 'True' on a new line, indicating the second version of the function is the correct fix.

```
assignment_7.5.py > ...
1 # Bug: Floating point precision issue
2 def check_sum():
3     return (0.1 + 0.2) == 0.3
4
5 print(check_sum())
6
7 # Fix the above code
8 def check_sum():
9     return abs((0.1 + 0.2) - 0.3) < 1e-9
10
11 print(check_sum())
12
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"
False
True
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

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.

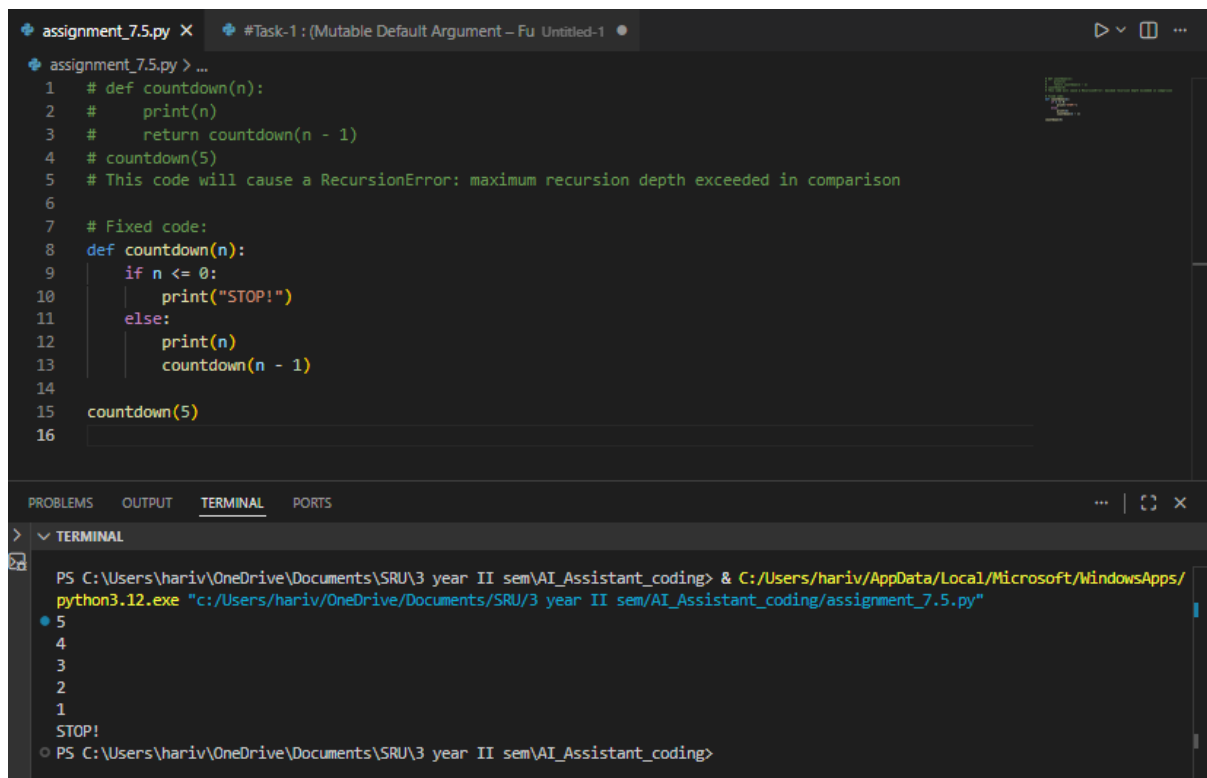
Task 3: Recursion Error – Missing Base Case

Prompt:

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

#Fixed Code:

Code & Output:



The screenshot shows a code editor with two tabs: 'assignment_7.5.py' and '#Task-1: (Mutable Default Argument - Fu Untitled-1)'. The Python script in the first tab defines a recursive function 'countdown(n)'. The initial version (lines 1-5) is commented out and includes a note: '# This code will cause a RecursionError: maximum recursion depth exceeded in comparison'. The 'Fixed code' (lines 8-15) includes a base case: 'if n <= 0: print("STOP!")'. The function is called with 'countdown(5)'. The terminal window at the bottom shows the command to run the script, followed by the output: '5', '4', '3', '2', '1', and 'STOP!'.

```
assignment_7.5.py > ...
1 # def countdown(n):
2 #     print(n)
3 #     return countdown(n - 1)
4 # countdown(5)
5 # This code will cause a RecursionError: maximum recursion depth exceeded in comparison
6
7 # Fixed code:
8 def countdown(n):
9     if n <= 0:
10         print("STOP!")
11     else:
12         print(n)
13         countdown(n - 1)
14
15 countdown(5)
16
```

PROBLEMS OUTPUT **TERMINAL** PORTS

PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"

5
4
3
2
1
STOP!
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>

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:



The screenshot shows a VS Code editor with two tabs: 'assignment_7.5.py' and '#Task-1: (Mutable Default Argument – Fu Untitled-1)'. The 'assignment_7.5.py' tab is active, displaying a Python script. The script defines a function 'get_value()' that attempts to access a non-existent key 'c' in a dictionary. The terminal at the bottom shows the command to run the script, which results in a 'KeyError: 'c''.

```
assignment_7.5.py > get_value
1 # def get_value():
2 #     data = {"a": 1, "b": 2}
3 #     return data["c"]
4 # print(get_value())
5 # This code will cause a KeyError
6
7 # Fixed code:
8 # Bug: Accessing non-existing key
9 def get_value():
10     data = {"a": 1, "b": 2}
11     return data.get("c", "Key not found")
12
13 print(get_value())
14
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"
Key not found
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

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.

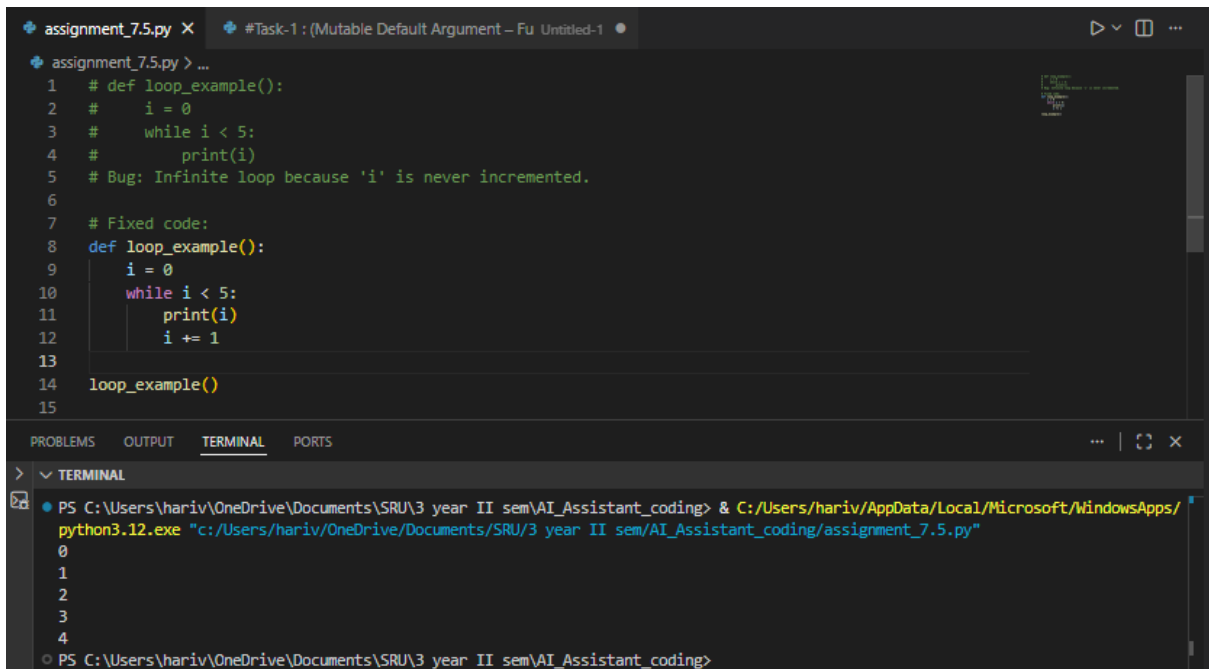
Task 5: Infinite Loop – Wrong Condition

Prompt:

#Bug: Infinite loop because 'i' is never incremented.

#Fixed Code:

Code & Output:



The screenshot shows a code editor with two tabs: 'assignment_7.5.py' and '#Task-1: (Mutable Default Argument – Fu Untitled-1)'. The 'assignment_7.5.py' tab is active, displaying a Python script. The script defines a function 'loop_example()' with a 'while' loop. The initial code (lines 1-6) has a bug where the loop variable 'i' is never incremented, leading to an infinite loop. The fixed code (lines 7-15) adds an increment step 'i += 1' to the loop. The terminal window at the bottom shows the command 'python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"' and the output '0', '1', '2', '3', '4'.

```
assignment_7.5.py > ...
1 # def loop_example():
2 #     i = 0
3 #     while i < 5:
4 #         print(i)
5 # Bug: Infinite loop because 'i' is never incremented.
6
7 # Fixed code:
8 def loop_example():
9     i = 0
10    while i < 5:
11        print(i)
12        i += 1
13
14    loop_example()
15
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"

0
1
2
3
4

PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>

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.

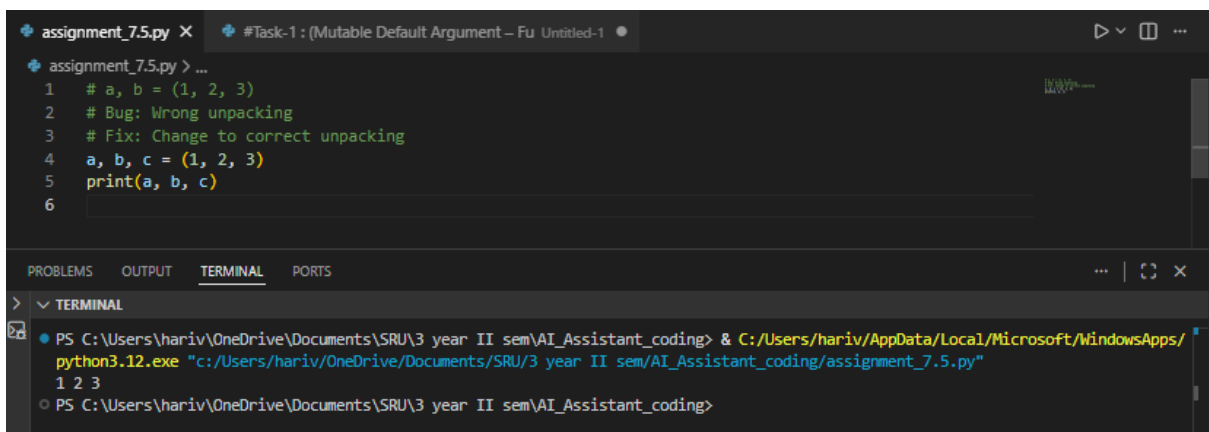
Task 6: Unpacking Error – Wrong Variables

Prompt:

#Bug: Wrong unpacking

#Fix : Change to correct unpacking

Code & Output:



The screenshot shows a code editor with two tabs: 'assignment_7.5.py' and '#Task-1: (Mutable Default Argument – Fu Untitled-1)'. The 'assignment_7.5.py' tab is active, displaying a Python script. The script defines a tuple 'a, b = (1, 2, 3)' and attempts to unpack it into three variables 'a, b, c'. The initial code (lines 1-3) has a bug where the unpacking is incorrect. The fixed code (lines 4-6) changes the unpacking to 'a, b, c = (1, 2, 3)'. The terminal window at the bottom shows the command 'python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"' and the output '1 2 3'.

```
assignment_7.5.py > ...
1 # a, b = (1, 2, 3)
2 # Bug: Wrong unpacking
3 # Fix: Change to correct unpacking
4 a, b, c = (1, 2, 3)
5 print(a, b, c)
6
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"

1 2 3

PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>

Explanation:

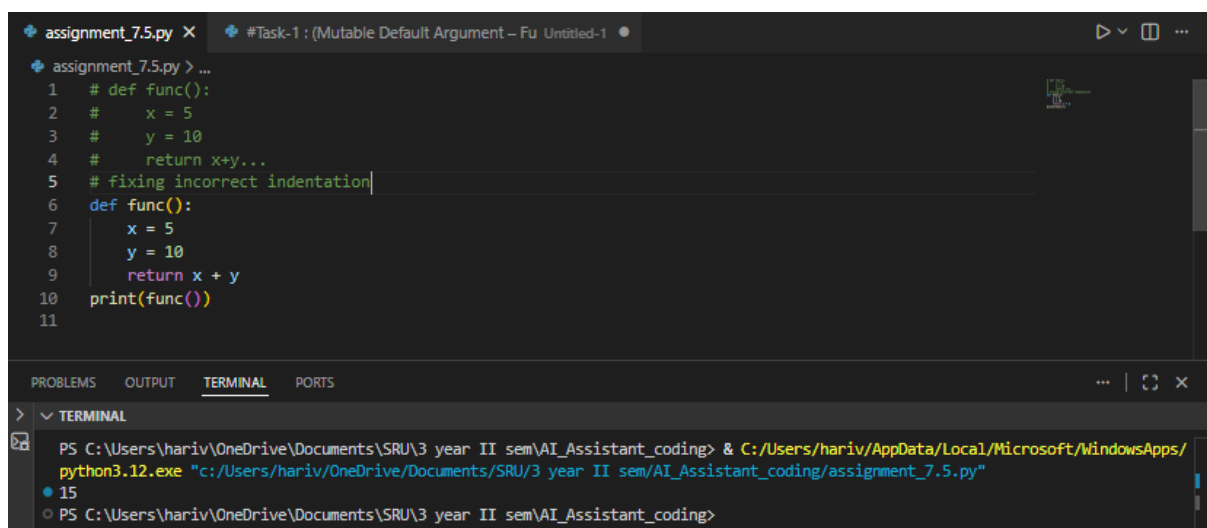
The AI identified that there was a mismatch between the number of variables and the values being unpacked from the tuple. It suggested fixing this either by adding more variables or by using a placeholder variable (_) for values that are not needed. This approach allows the tuple to be unpacked correctly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

Task 7: Mixed Indentation – Tabs vs Spaces

Prompt:

#Fixing incorrect indentation

Code & Output:



```
assignment_7.5.py X #Task-1 : (Mutable Default Argument – Fu Untitled-1
1 # def func():
2 #     x = 5
3 #     y = 10
4 #     return x+y...
5 # fixing incorrect indentation
6 def func():
7     x = 5
8     y = 10
9     return x + y
10 print(func())
11

PROBLEMS OUTPUT TERMINAL PORTS
> TERMINAL
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"
15
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

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.

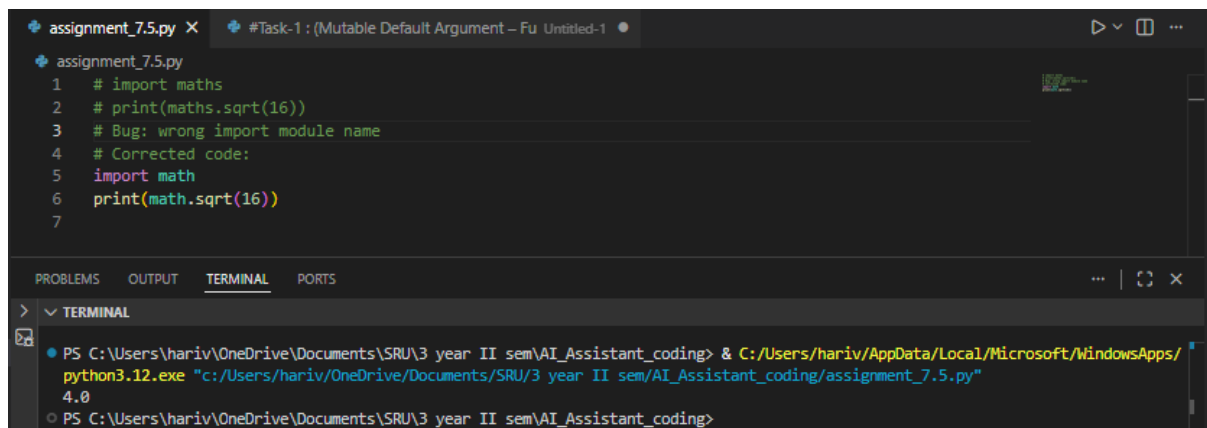
Task 8: Import Error – Wrong Module Usage

Prompt:

#Bug: Wrong import module name

#Corrected Code:

Code & Output:



The screenshot shows a code editor with two tabs: 'assignment_7.5.py' and '#Task-1 : (Mutable Default Argument - Fu Untitled-1)'. The 'assignment_7.5.py' tab is active, displaying the following code:

```
1 # import maths
2 # print(maths.sqrt(16))
3 # Bug: wrong import module name
4 # Corrected code:
5 import math
6 print(math.sqrt(16))
7
```

Below the code editor is a terminal window with tabs for 'PROBLEMS', 'OUTPUT', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active, showing the command to run the script and its output:

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_7.5.py"
4.0
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

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.