

AI Assisted Coding

Assignment 7.5

Name:R.Bharadwaj

Hall ticket no: 2303A51610

Batch no: 19

Task 1:

Prompt:

Fix the following Python code that has a mutable default argument bug.

Rewrite the function so that each function call gets a new list.

Show the corrected code and its output.

Code:

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

```
items.append(item)
```

```
return items
```

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

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

Code and Output:

```
17 # Bug: Mutable default argument - FIXED
18 def add_item(item, items=None):
19     if items is None:
20         items = []
21         items.append(item)
22     return items
23
24 print(add_item(1))
25 print(add_item(2))
26
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS GITLENS AZURE

Python Debug Console

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\vscode\extensions\ms-python.debugpy-2025.19.2026012701-win32-x64\bundled\libs\debugpy\launcher' '64789' '--' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 7.5.py'
[1]
[2]
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

Explanation:

The given code uses a mutable default argument `items=[]`, so the same list is reused across function calls. Because of this, the first call `add_item(1)` outputs `[1]`, but the second call `add_item(2)` incorrectly outputs `[1, 2]` instead of a new list. To fix this, the default argument is changed to `None`, and a new list is created inside the function when `items` is `None`. After fixing, each function call gets a

fresh list, so the outputs become [1] and [2], which matches the prompt's requirement and correctly removes the mutable default argument bug.

Task 2:

Prompt:

Fix the following Python code where a floating-point comparison fails.

Rewrite the function using a tolerance-based comparison so the result is correct. Show the corrected code and its output.

Code:

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

Code and Output:

The image shows a Visual Studio Code (VS Code) editor window with a Python file named 'Assignment 7.5.py'. The code in the editor is as follows:

```
44 # Bug: Floating point precision issue - FIXED
45 import math
46
47 def check_sum():
48     return math.isclose(0.1 + 0.2, 0.3)
49
50 print(check_sum())
51
52
```

The editor's interface includes a sidebar on the right with a 'Run and Debug' icon and a 'Python Debug Console' icon. Below the editor, the 'TERMINAL' tab is active, showing the command prompt output:

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 7.5.py'
True
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

The bottom of the image shows the Windows taskbar with the following icons: Screen Reader Optimized, In 46, Col 1, Tab Size: 4, UTF-8, CRLF, Python, 3.13.9 (Microsoft Store), Go Live, and Prettier.

Explanation:

The given code directly compares floating-point values using `==`, so `(0.1 + 0.2) == 0.3` evaluates to `False` due to floating-point precision errors. As a result, `check_sum()` prints `False`, which is incorrect for the intended logic. To fix this, the function is rewritten using a tolerance-based comparison with `math.isclose()`, which checks whether the two values are close enough within an acceptable margin. After applying the fix, `check_sum()` correctly returns and prints `True`, satisfying the prompt's requirement.

Task 3:

Prompt:

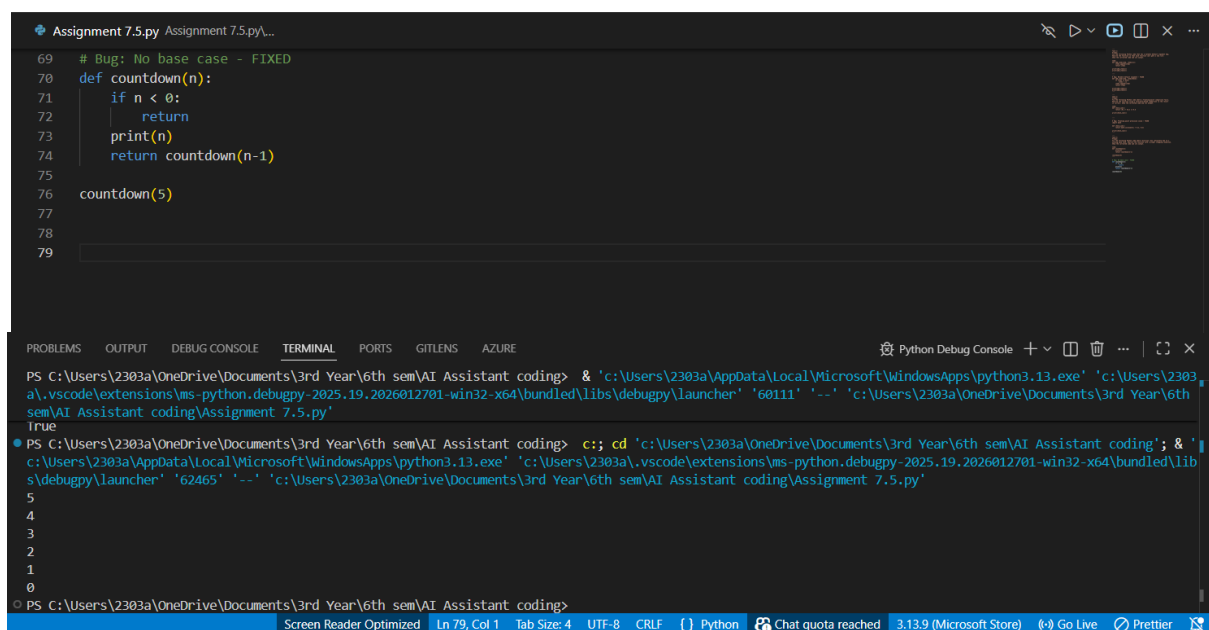
Fix the following Python code where recursion runs infinitely due to a missing base case. Rewrite the function with a proper stopping condition.

Show the corrected code and its output.

Code:

```
def countdown(n):  
    print(n)  
    return countdown(n-1)  
  
countdown(5)
```

Code and Output:



The screenshot shows a VS Code editor window with a file named 'Assignment 7.5.py'. The code in the editor is as follows:

```
69 # Bug: No base case - FIXED  
70 def countdown(n):  
71     if n < 0:  
72         return  
73     print(n)  
74     return countdown(n-1)  
75  
76 countdown(5)  
77  
78  
79
```

The terminal output at the bottom shows the execution of the code:

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 7.5.py'  
True  
5  
4  
3  
2  
1  
0  
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

The terminal output indicates that the code executed successfully, printing the numbers 5, 4, 3, 2, 1, and 0, and returning `True`.

Explanation:

The original code causes infinite recursion because it calls `countdown(n-1)` without any base case to stop the function, eventually leading to a recursion error. In the corrected version, a proper stopping condition `if n < 0: return` is added, which halts the recursion once the value of `n` goes below zero. After fixing the code, calling `countdown(5)` correctly prints the numbers 5 4 3 2 1 0 and then stops, fulfilling the prompt's requirement.

Task 4:

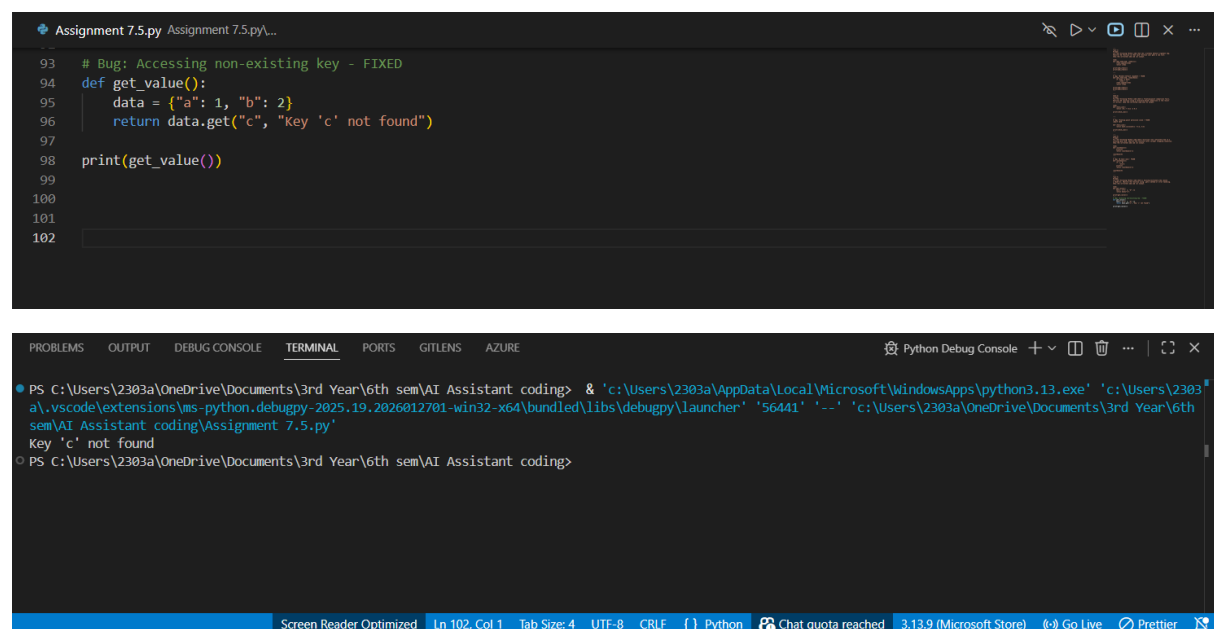
Prompt:

Fix the following Python code where a missing dictionary key causes a `KeyError`. Rewrite the function using `.get()` method or error handling. Show the corrected code and its output.

Code:

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

Code and Output:



The screenshot shows a VS Code editor window with a file named 'Assignment 7.5.py'. The code in the editor is as follows:

```
93 # Bug: Accessing non-existing key - FIXED  
94 def get_value():  
95     data = {"a": 1, "b": 2}  
96     return data.get("c", "Key 'c' not found")  
97  
98 print(get_value())  
99  
100  
101  
102
```

Below the editor, the terminal window shows the output of the code:

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 7.5.py'  
Key 'c' not found  
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

Explanation:

The original problem is accessing a dictionary key that does not exist, which would normally raise a `KeyError`. In the corrected code, the `get()` method is used instead of direct key access, providing a default message when the key `"c"` is missing. As a result, the function safely returns `"Key 'c' not found"`.

found" instead of crashing. When `get_value()` is called, the output is the message indicating the key is not present, correctly fixing the issue as required by the prompt.

Task 5:

Prompt:

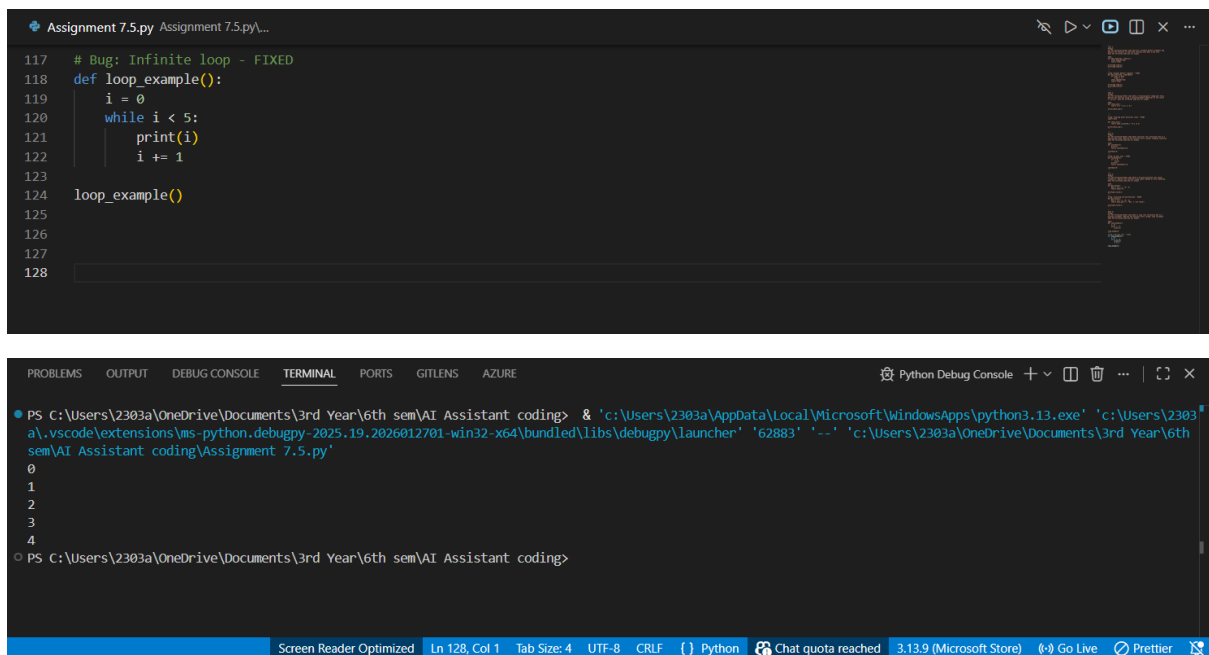
Fix the following Python code where a missing dictionary key causes a `KeyError`. Rewrite the function using `.get()` method or error handling.

Show the corrected code and its output.

Code:

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

Code and Output:



The screenshot displays the Visual Studio Code editor with a Python file named `Assignment 7.5.py`. The code defines a function `loop_example()` that initializes `i = 0` and enters a `while i < 5:` loop. Inside the loop, it prints `i` and increments `i` by 1. The function is then called. The terminal output shows the execution of the script, which prints the numbers 0, 1, 2, 3, and 4, confirming that the loop terminates correctly after the fix.

```
117 # Bug: Infinite loop - FIXED  
118 def loop_example():  
119     i = 0  
120     while i < 5:  
121         print(i)  
122         i += 1  
123  
124     loop_example()  
125  
126  
127  
128
```

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\vscode\extensions\ms-python.debugpy-2025.19.2026012701-win32-x64\bundled\libs\debugpy\launcher' '62883' '--' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 7.5.py'  
0  
1  
2  
3  
4  
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

Explanation:

The original code enters an infinite loop because the loop variable `i` is never incremented, so the condition `i < 5` is always true. In the corrected version, `i += 1` is added inside the while loop, allowing `i` to increase on each iteration and eventually break the loop. After fixing the code, calling `loop_example()` correctly prints 0 1 2 3 4 and then terminates, satisfying the prompt's requirement.

Task 6:

Prompt:

Fix the following Python code where tuple unpacking fails.

Rewrite the function using correct unpacking or using `_` for extra values.

Show the corrected code and its output.

Code:

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

Code and Output:

```
Assignment 7.5.py Assignment 7.5.py\...
139 # Bug: Wrong unpacking - FIXED
140 a, b, c = (1, 2, 3)
141 print(a, b, c)
142
143 # Alternative: Using _ to ignore extra values
144 a, b, _ = (1, 2, 3)
145 print(a, b)
146
147
148
149
150
```

Explanation:

The original code fails because it tries to unpack three values (1, 2, 3) into only two variables a and b, which causes a tuple unpacking error. In the corrected version, the unpacking is fixed either by matching the number of variables (a, b, c = (1, 2, 3)) or by using _ to ignore the extra value (a, b, _ = (1, 2, 3)). After applying the fix, the code runs successfully and prints 1 2 3 for the first case and 1 2 for the second, correctly resolving the issue as required by the prompt.

Task 7:

Prompt:

Fix the following Python code where mixed indentation breaks execution.

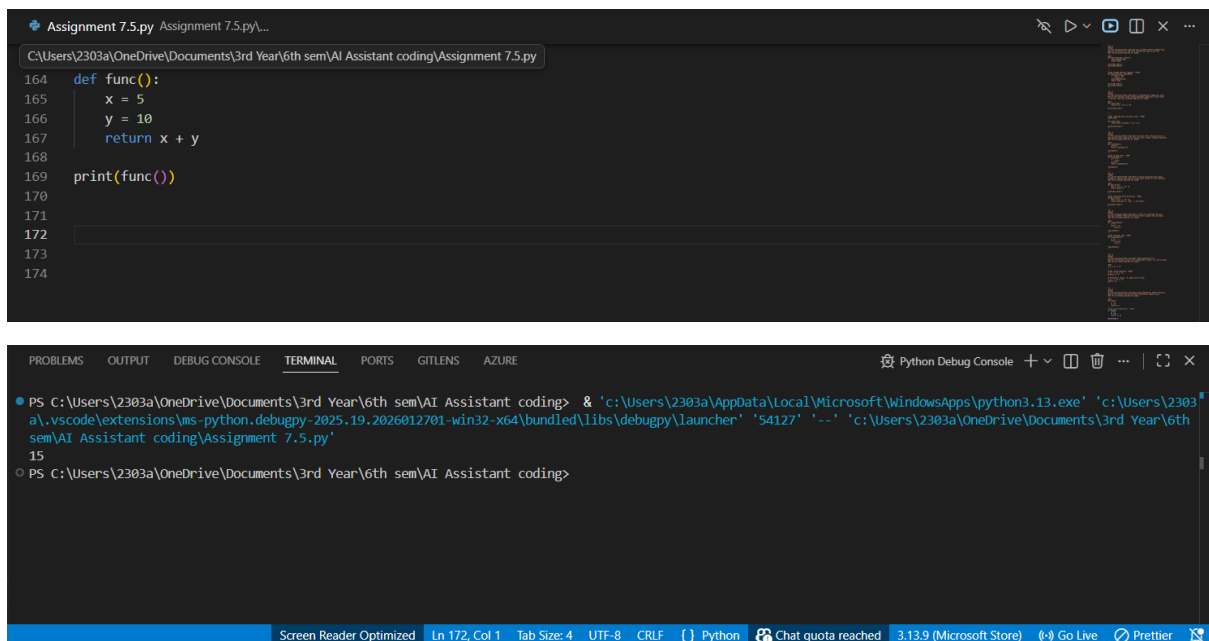
Rewrite the function using consistent indentation (spaces only).

Show the corrected code and its output.

Code:

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

Code and Output:



The screenshot shows a VS Code editor window with a file named 'Assignment 7.5.py'. The code in the editor is as follows:

```
164 def func():  
165     x = 5  
166     y = 10  
167     return x + y  
168  
169 print(func())  
170  
171  
172  
173  
174
```

Below the editor, the terminal window shows the command to run the script and its output:

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 7.5.py'  
15  
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

The terminal output shows the number 15, which is the sum of x and y.

Explanation:

The original code fails because it uses mixed indentation, which breaks Python's block structure and causes a syntax or indentation error. In the corrected version, consistent spacing (spaces only) is used for all lines inside the function, and the expression is properly written as `x + y`. After fixing the indentation and syntax, calling `func()` correctly returns the sum of the two variables, and the output printed is 15, satisfying the prompt's requirement.

Task 8:

Prompt:

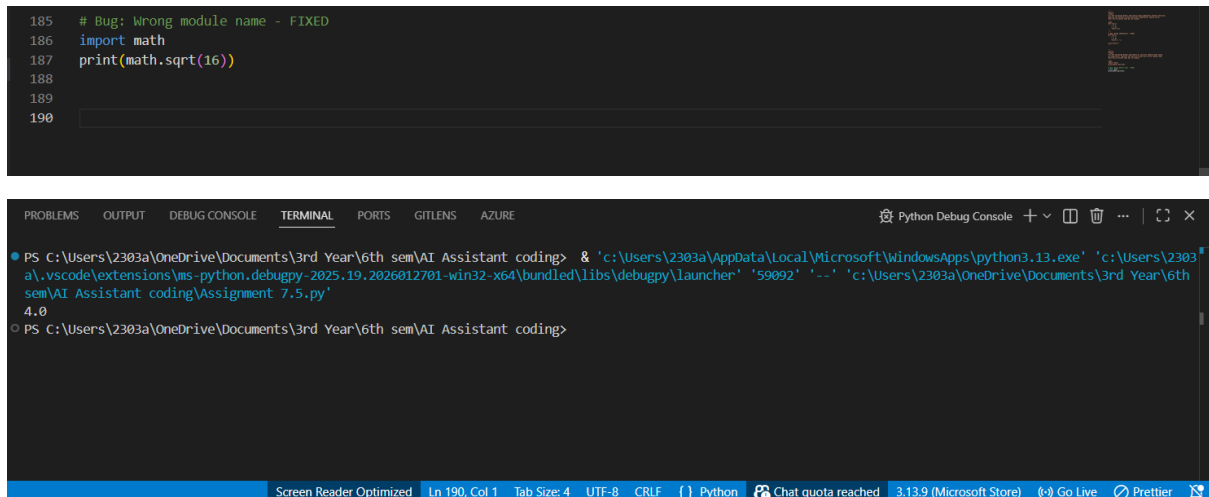
Fix the following Python code where an incorrect module name causes an ImportError. Rewrite the function using the correct module name.

Show the corrected code and its output.

Code:

```
import maths  
  
print(maths.sqrt(16))
```

Code and Explanation:



The image shows a code editor window with a Python file. The code is as follows:

```
185 # Bug: Wrong module name - FIXED  
186 import math  
187 print(math.sqrt(16))  
188  
189  
190
```

Below the code editor is a terminal window. The terminal shows the command prompt running the code, and the output is 4.0.

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\vscode\extensions\ms-python.debugpy-2025.19.2026012701-win32-x64\bundle\libs\debugpy\launcher' '59092' '--' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 7.5.py'  
4.0  
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

Explanation:

The original code raises an ImportError because it tries to import a non-existent module named `maths`. In the corrected version, the proper built-in module name `math` is used, allowing the `sqrt()` function to execute correctly. After fixing the module name, the program runs successfully and prints 4.0 as the output, which satisfies the prompt's requirement.