AI Assisted Coding

Assignment 7.5

Name: P. Vineeth Kumar

Hall ticket no: 2303A51256

Batch no: 19

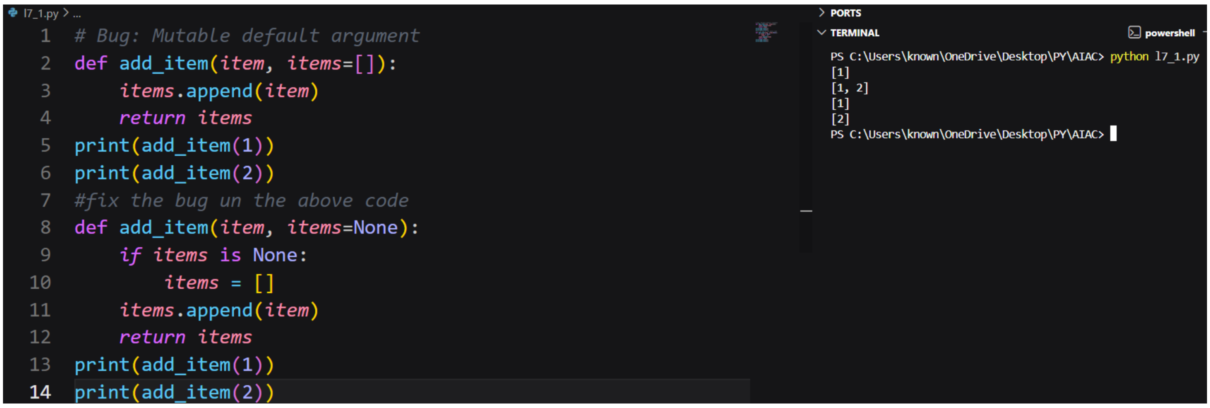
Task 1: Mutable Default Argument – Function Bug

Prompt:

#Bug: Mutable default argument

#Fix the bug in the above code

Code & Output:



```python
# Bug: Mutable default argument
def add_item(item, items=[]):
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
#fix the bug un the above code
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
[1]
[1, 2]
[1]
[2]
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

The AI correctly identified that using a mutable object (list) as a default argument leads to shared state across function calls. This results in unexpected accumulation of values. The AI-fixed version initializes the list inside the function when no argument is provided, ensuring a fresh list is created each time. This approach prevents side effects and follows best practices in Python function design.

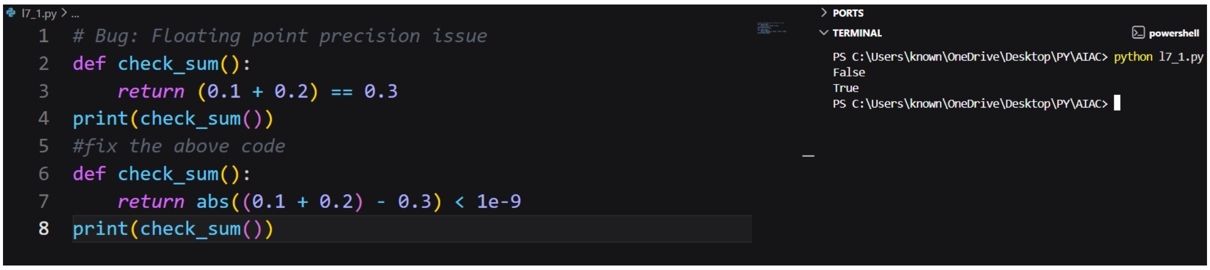
Task 2: Floating-Point Precision Error

Prompt:

#Bug: Floating point precision issue

#Fix the above code

Code & Output:



```python
# Bug: Floating point precision issue
def check_sum():
    return (0.1 + 0.2) == 0.3
print(check_sum())
#fix the above code
def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-9
print(check_sum())
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
False
True
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

The AI identified that floating-point numbers cannot always be compared directly due to precision limitations in binary representation. Instead of using equality comparison, the AI suggested checking whether the difference between values lies within an acceptable tolerance. This solution improves reliability and accuracy in numerical computations.

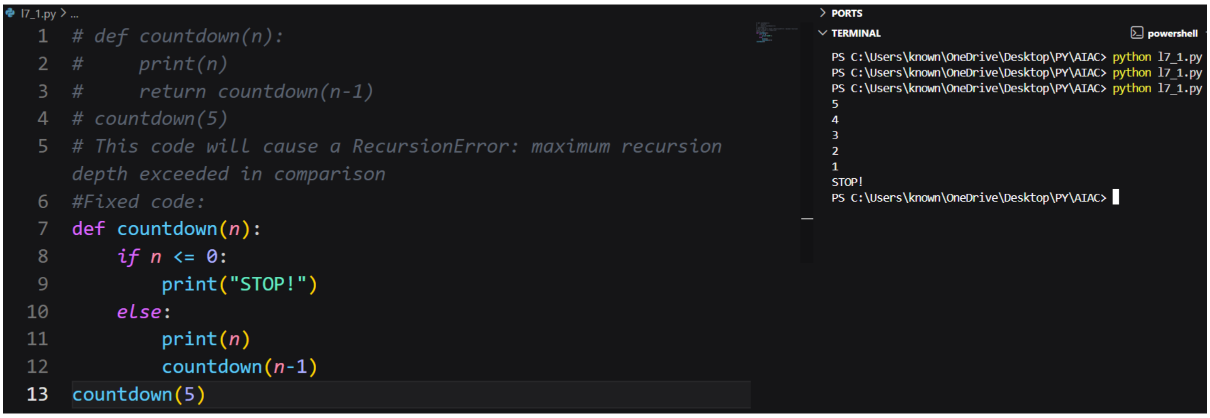
Task 3: Recursion Error – Missing Base Case

Prompt:

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

#Fixed Code:

Code & Output:



```python
# def countdown(n):
#     print(n)
#     return countdown(n-1)
# countdown(5)
# This code will cause a RecursionError: maximum recursion depth exceeded in comparison
#Fixed code:
def countdown(n):
    if n <= 0:
        print("STOP!")
    else:
        print(n)
        countdown(n-1)
countdown(5)
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
5
4
3
2
1
STOP!
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

The AI detected that the recursive function lacks a stopping condition, causing infinite recursion and stack overflow. By adding a proper base case, the AI ensured that recursion terminates correctly. This fix demonstrates the importance of defining exit conditions in recursive algorithms.

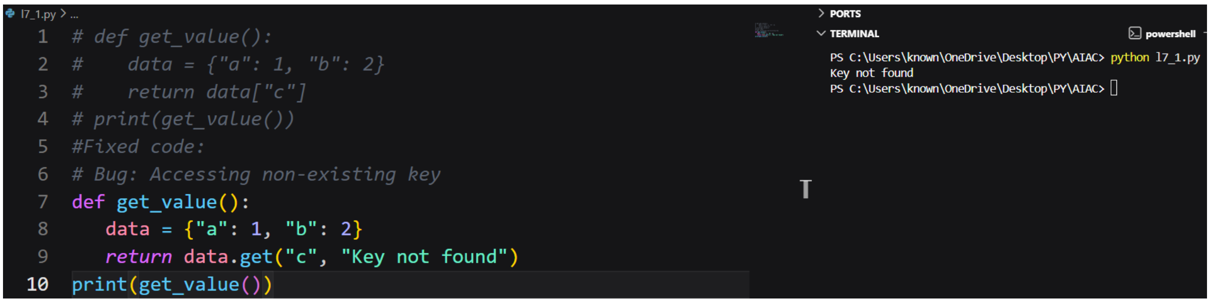
Task 4: Dictionary Key Error

Prompt:

#Fixed Code:

#Bug: Accessing non-existing key

Code & Output:



```python
# def get_value():
#    data = {"a": 1, "b": 2}
#    return data["c"]
# print(get_value())
#Fixed code:
# Bug: Accessing non-existing key
def get_value():
   data = {"a": 1, "b": 2}
   return data.get("c", "Key not found")
print(get_value())
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
Key not found
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> 
```

Explanation:

The AI recognized that directly accessing a missing dictionary key raises a KeyError. It suggested safer alternatives such as using the .get() method or handling the exception. This improves program robustness and prevents unexpected crashes during execution.

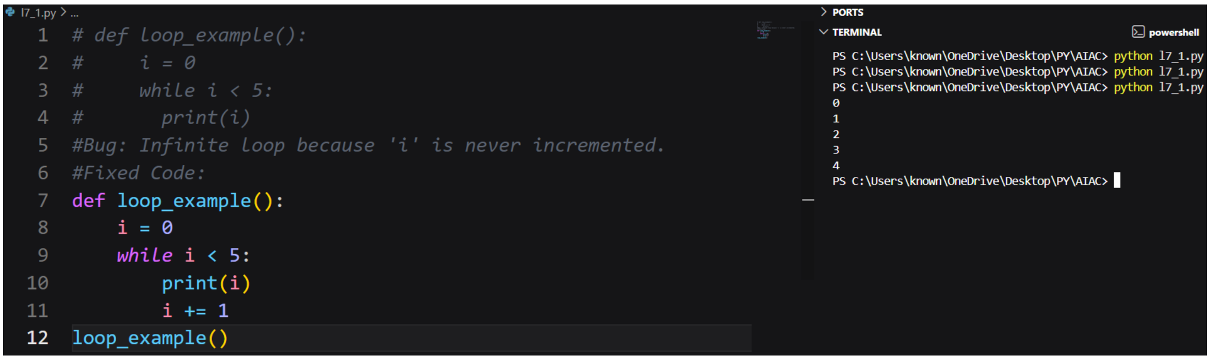
Task 5: Infinite Loop – Wrong Condition

Prompt:

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

#Fixed Code:

Code & Output:



```python
# def loop_example():
#     i = 0
#     while i < 5:
#         print(i)
#Bug: Infinite loop because 'i' is never incremented.
#Fixed Code:
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1
loop_example()
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
0
1
2
3
4
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> 
```

Explanation:

The AI identified that the loop variable was never updated, causing an infinite loop. By adding the correct increment operation, the loop now progresses toward its termination condition. This fix highlights the importance of updating loop control variables correctly.

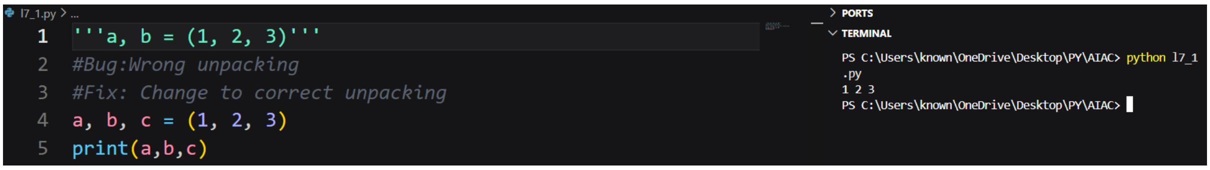
Task 6: Unpacking Error – Wrong Variables

Prompt:

#Bug: Wrong unpacking

#Fix : Change to correct unpacking

Code:



```python
'''a, b = (1, 2, 3)'''
#Bug:Wrong unpacking
#Fix: Change to correct unpacking
a, b, c = (1, 2, 3)
print(a,b,c)
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1
.py
1 2 3
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> 
```

Explanation:

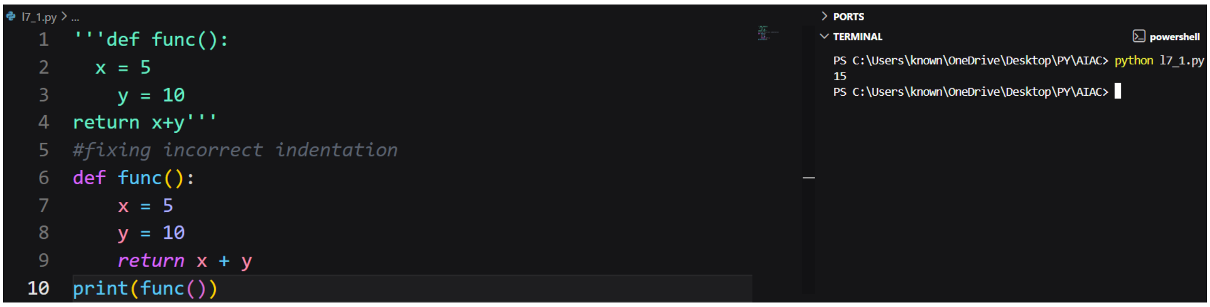
The AI detected a mismatch between the number of variables and values during tuple unpacking. It suggested either increasing the number of variables or using a placeholder variable (_) for unused values. This solution ensures correct unpacking without runtime errors.

Task 7: Mixed Indentation – Tabs vs Spaces

Prompt:

#Fixing incorrect indentation

Code:



```python
'''def func():
  x = 5
    y = 10
return x+y'''
#fixing incorrect indentation
def func():
    x = 5
    y = 10
    return x + y
print(func())
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
15
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> 
```

Explanation:

The AI identified inconsistent indentation as the root cause of the error. Python relies strictly on indentation for block definition. By applying consistent spacing throughout the function, the AI restored correct program execution and improved code readability.

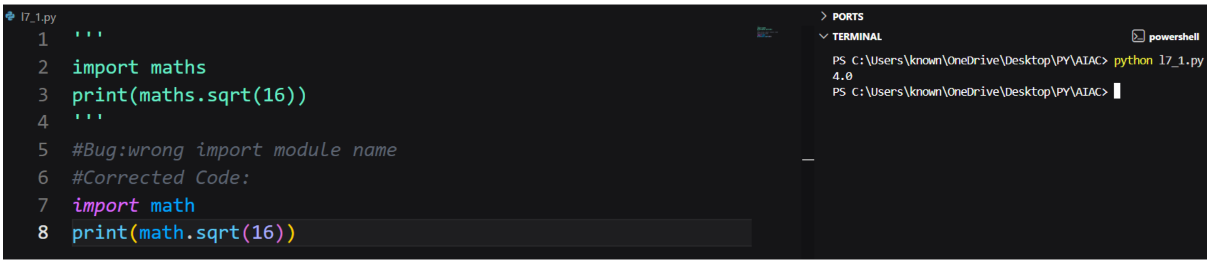
Task 8: Import Error – Wrong Module Usage

Prompt:

#Bug: Wrong import module name

#Corrected Code:

Code:



```
'''
import maths
print(maths.sqrt(16))
'''
#Bug:wrong import module name
#Corrected Code:
import math
print(math.sqrt(16))
```

```
PS C:\Users\known\OneDrive\Desktop\PY\AIAC> python l7_1.py
4.0
PS C:\Users\known\OneDrive\Desktop\PY\AIAC>
```

Explanation:

The AI recognized that the imported module name was incorrect. It suggested importing the standard Python math module instead. This fix resolves the import error and allows the program to use mathematical functions correctly.