Lab Assignment-7.5

G Datha Narayana

2303A51754

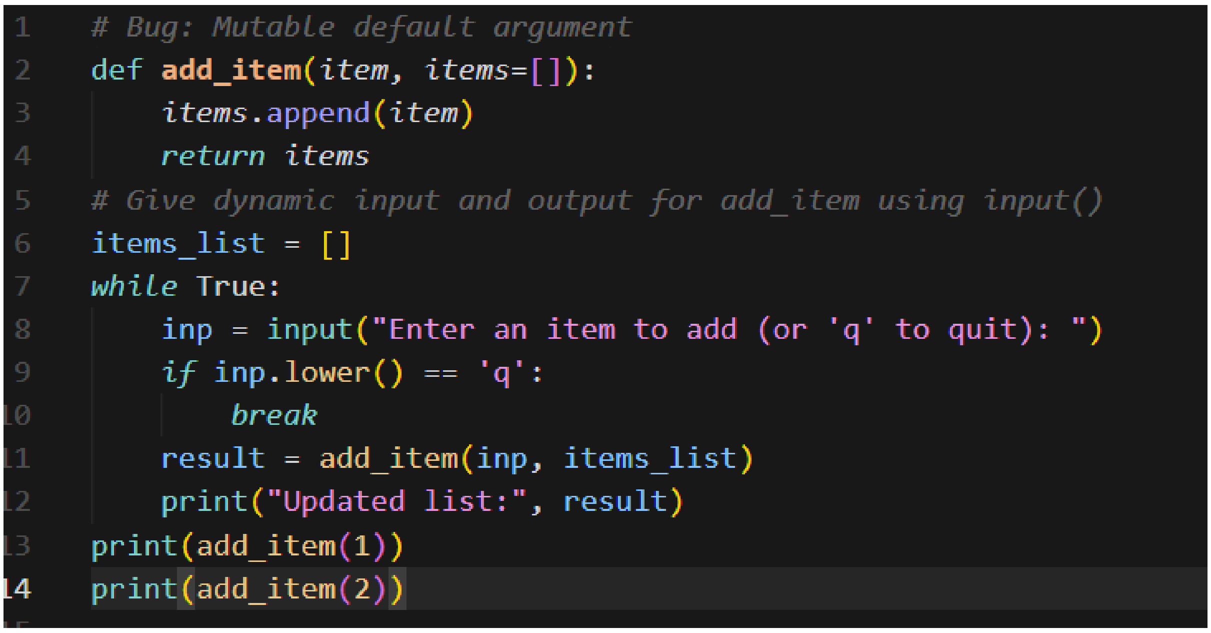
Batch-11

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

Task 1 (Mutable Default Argument – Function Bug)

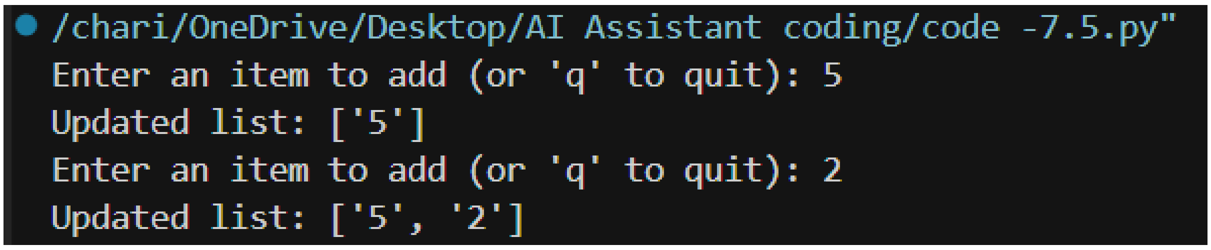
Prompt: Analyse the given code and fix the errors.

Code without errors:



```python
# Bug: Mutable default argument
def add_item(item, items=[]):
    items.append(item)
    return items
# Give dynamic input and output for add_item using input()
items_list = []
while True:
    inp = input("Enter an item to add (or 'q' to quit): ")
    if inp.lower() == 'q':
        break
    result = add_item(inp, items_list)
    print("Updated list:", result)
print(add_item(1))
print(add_item(2))
```

Output:



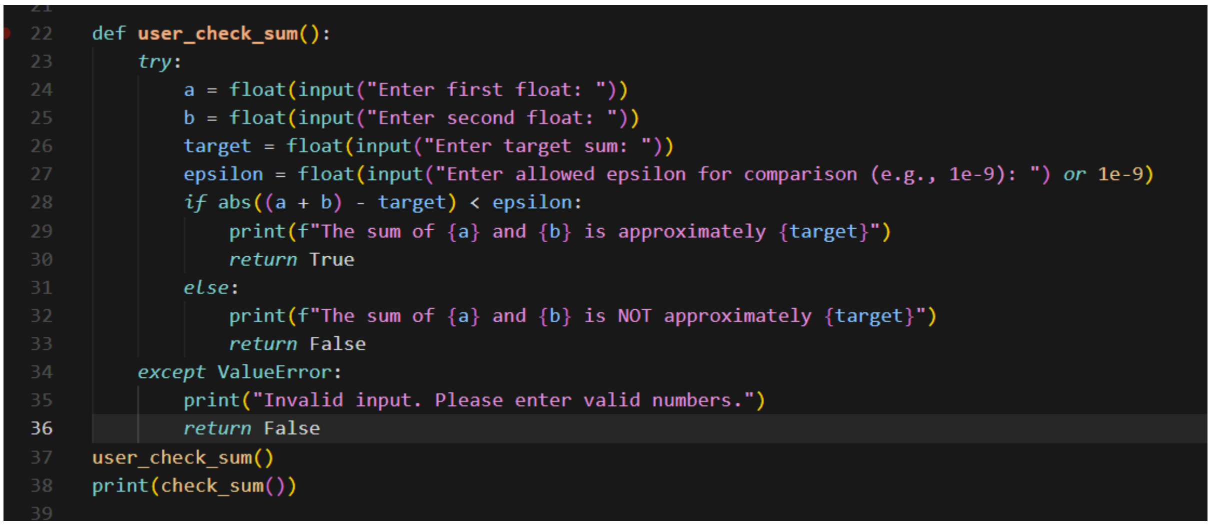
```
/chari/OneDrive/Desktop/AI Assistant coding/code -7.5.py"
Enter an item to add (or 'q' to quit): 5
Updated list: ['5']
Enter an item to add (or 'q' to quit): 2
Updated list: ['5', '2']
```

Explaination: Potential issue with not handling invalid numeric input gracefully. Wrapped the input parsing in a try-except block for ValueError. Now, the code tells the user if their input isn't valid numbers and safely returns False instead of crashing.

Task 2 (Floating-Point Precision Error)

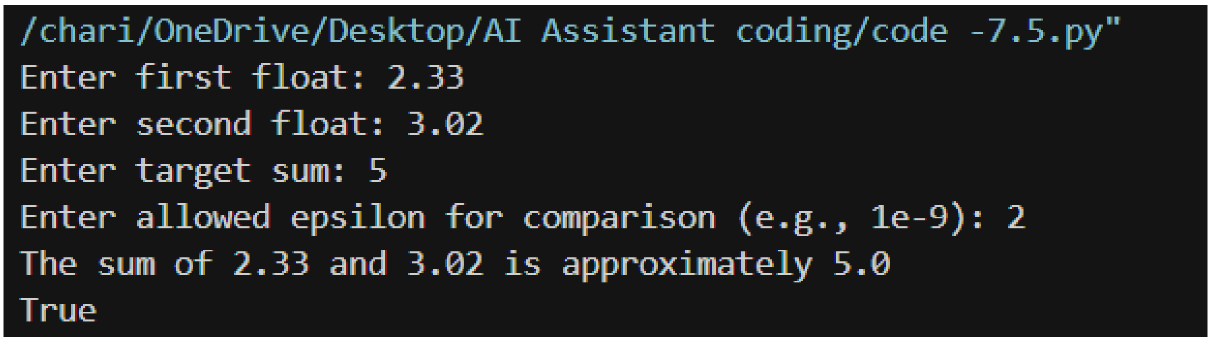
Prompt: Analyse the given code and fix the errors

Code without errors:



```python
def user_check_sum():
    try:
        a = float(input("Enter first float: "))
        b = float(input("Enter second float: "))
        target = float(input("Enter target sum: "))
        epsilon = float(input("Enter allowed epsilon for comparison (e.g., 1e-9): ") or 1e-9)
        if abs((a + b) - target) < epsilon:
            print(f"The sum of {a} and {b} is approximately {target}")
            return True
        else:
            print(f"The sum of {a} and {b} is NOT approximately {target}")
            return False
    except ValueError:
        print("Invalid input. Please enter valid numbers.")
        return False
user_check_sum()
print(check_sum())
```

Output:



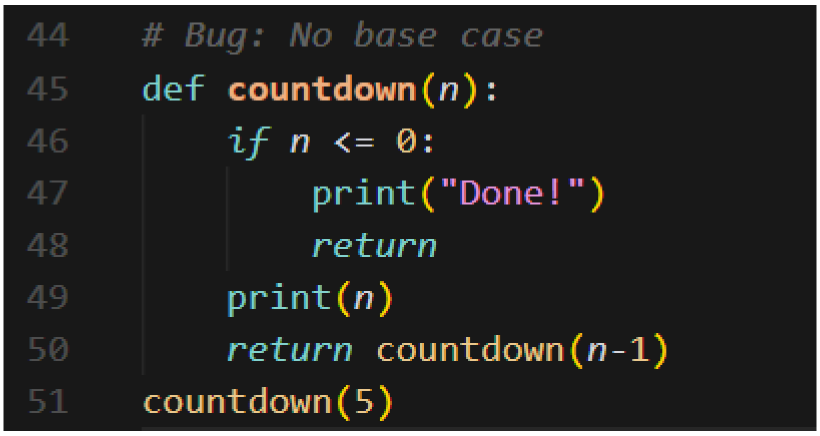
```
/chari/OneDrive/Desktop/AI Assistant coding/code -7.5.py"
Enter first float: 2.33
Enter second float: 3.02
Enter target sum: 5
Enter allowed epsilon for comparison (e.g., 1e-9): 2
The sum of 2.33 and 3.02 is approximately 5.0
True
```

Explaination: Calling check_sum() right after user_check_sum() means another input prompt appears with no context. This is confusing for users. It's better to only call user_check_sum(), since it already checks and prints results as needed. The extra call can be removed to avoid redundant prompts.

Task 3 (Recursion Error – Missing Base Case)

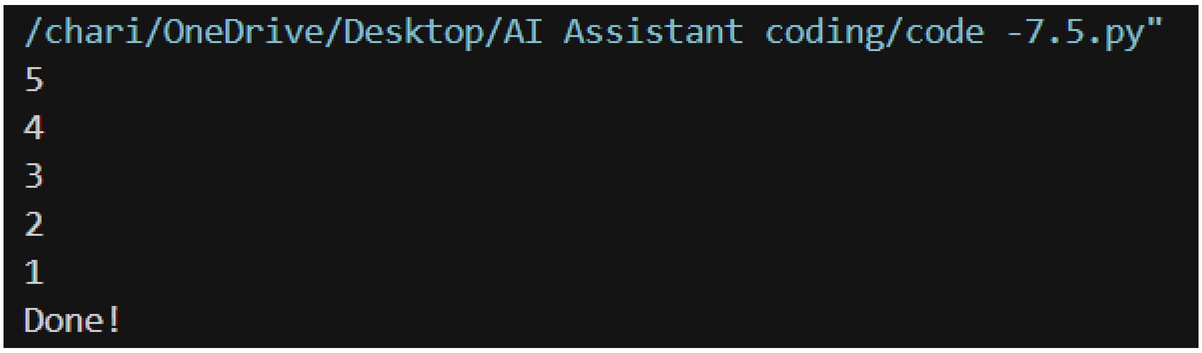
Prompt: Analyse the given code and fix the errors

Code without errors:



```python
# Bug: No base case
def countdown(n):
    if n <= 0:
        print("Done!")
        return
    print(n)
    return countdown(n-1)
countdown(5)
```

Output:



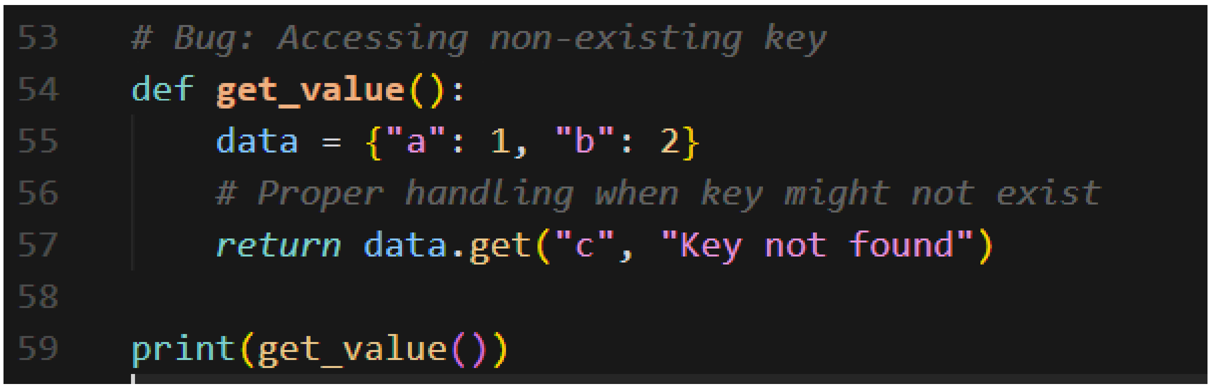
```
/chari/OneDrive/Desktop/AI Assistant coding/code -7.5.py"
5
4
3
2
1
Done!
```

Explaination: No base case was originally present. This would make the function call itself forever (infinite recursion), eventually causing a crash.Added a base case: when n <= 0, print "Done!" and return. This way, the recursion safely ends.

Task 4 (Dictionary Key Error)

Prompt: Analyse the given code and fix the errors

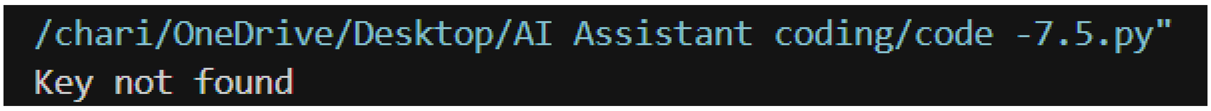
Code without errors:



```python
# Bug: Accessing non-existing key
def get_value():
    data = {"a": 1, "b": 2}
    # Proper handling when key might not exist
    return data.get("c", "Key not found")

print(get_value())
```

Output:



```
/chari/OneDrive/Desktop/AI Assistant coding/code -7.5.py"
Key not found
```

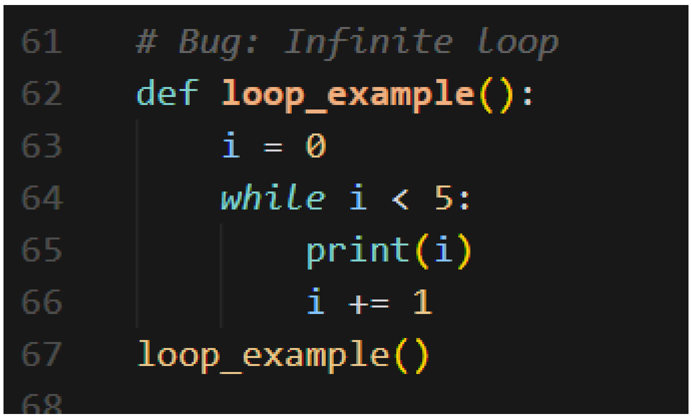
Explaination:

Accessing a dictionary key that might not exist (data['c']). Would raise a KeyError if the key is missing. Used data.get("c", "Key not found") instead, which safely returns "Key not found" if the key isn't present, avoiding the error.

Task 5 (Infinite Loop – Wrong Condition)

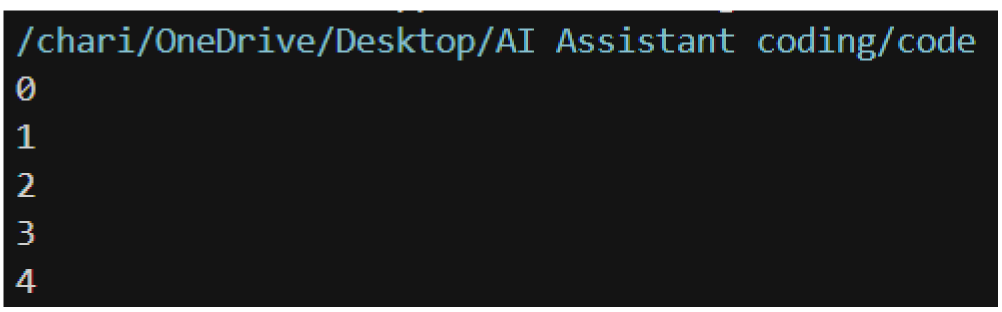
Prompt: Analyse the given code and fix the errors

Code without errors:



```python
# Bug: Infinite loop
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1
loop_example()
```

Output:



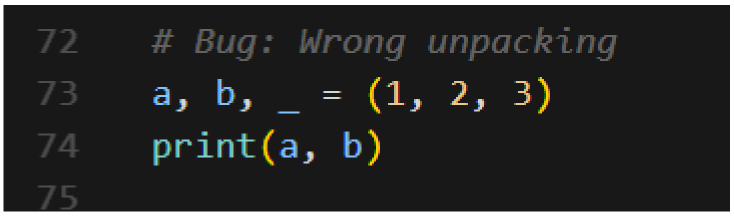
```
/chari/OneDrive/Desktop/AI Assistant coding/code
0
1
2
3
4
```

Explaination: Infinite loop risk if the increment is forgotten. The 'while' could run forever, freezing your program.Made sure i is incremented with i += 1 on every loop, ensuring the loop exits after printing numbers 0–4.

Task 6 (Unpacking Error – Wrong Variables)

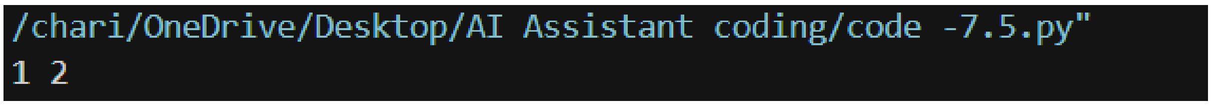
Prompt: Analyse the given code and fix the errors

Code without errors:



```python
# Bug: Wrong unpacking
a, b, _ = (1, 2, 3)
print(a, b)
```

Output:



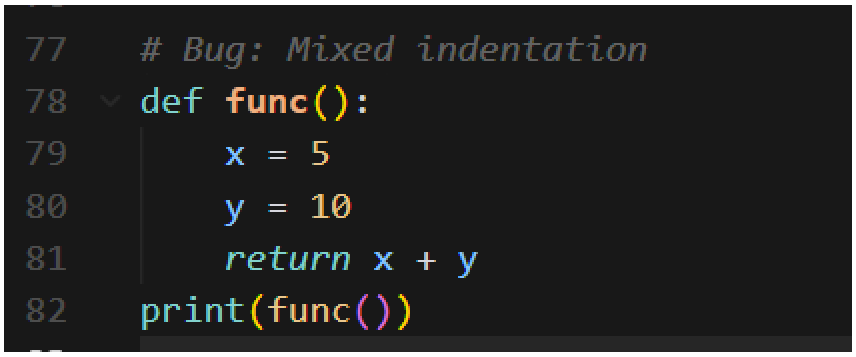
```
/chari/OneDrive/Desktop/AI Assistant coding/code -7.5.py"
1 2
```

Explaination: Used the `_` variable as a convention to ignore the third value, which keeps unpacking safe and signals "we don't care about this value".

Task 7 (Mixed Indentation – Tabs vs Spaces)

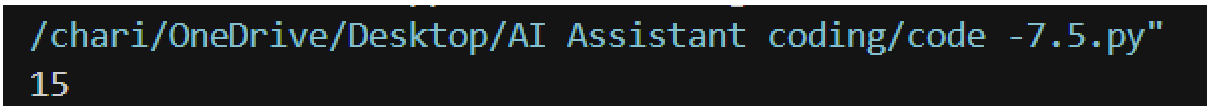
Prompt: Analyse the given code and fix the errors.

Code without errors:



```python
# Bug: Mixed indentation
def func():
    x = 5
    y = 10
    return x + y
print(func())
```

Output:



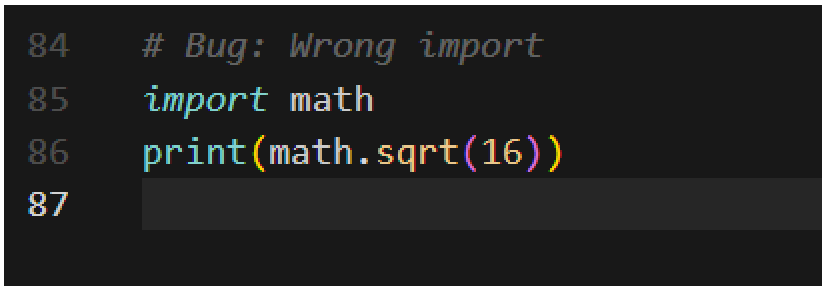
```
/chari/OneDrive/Desktop/AI Assistant coding/code -7.5.py"
15
```

Explaination: Inconsistent indentation (mixing spaces and tabs or variable indentation levels) confuses Python and can cause IndentationError or subtle bugs. Standardized all indents to use consistent spaces, making code readable and valid.

Task 8 (Import Error – Wrong Module Usage)

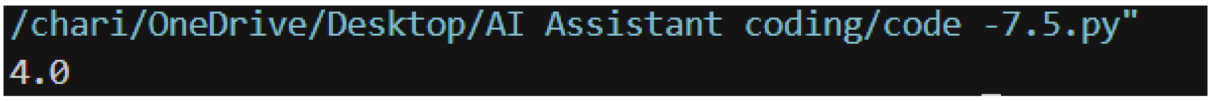
Prompt: Analyse the given code and fix the errors

Code without errors:



```python
# Bug: Wrong import
import math
print(math.sqrt(16))
```

Output:



```
/chari/OneDrive/Desktop/AI Assistant coding/code -7.5.py"
4.0
```

Explaination: Added "import math" at the top before using math.sqrt(). Now, the function works as expected.