

Lab Assignment-7.5

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

T Hrushikesh

2303A51788

Batch-12

Task 1 (Mutable Default Argument – Function Bug)

Prompt: Analyse the given code and fix the errors.

Code without errors:

```
1  # Bug: Mutable default argument
2  def add_item(item, items=[]):
3      items.append(item)
4      return items
5  # Give dynamic input and output for add_item using input()
6  items_list = []
7  while True:
8      inp = input("Enter an item to add (or 'q' to quit): ")
9      if inp.lower() == 'q':
10         break
11         result = add_item(inp, items_list)
12         print("Updated list:", result)
13     print(add_item(1))
14     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']
```

Explanation: 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:

```
21
22 def user_check_sum():
23     try:
24         a = float(input("Enter first float: "))
25         b = float(input("Enter second float: "))
26         target = float(input("Enter target sum: "))
27         epsilon = float(input("Enter allowed epsilon for comparison (e.g., 1e-9): ") or 1e-9)
28         if abs((a + b) - target) < epsilon:
29             print(f"The sum of {a} and {b} is approximately {target}")
30             return True
31         else:
32             print(f"The sum of {a} and {b} is NOT approximately {target}")
33             return False
34     except ValueError:
35         print("Invalid input. Please enter valid numbers.")
36         return False
37 user_check_sum()
38 print(check_sum())
39
```

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
```

Explanation: 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:

```
44  # Bug: No base case
45  def countdown(n):
46      if n <= 0:
47          print("Done!")
48          return
49      print(n)
50      return countdown(n-1)
51  countdown(5)
```

Output:

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

Explanation: 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 $n \leq 0$, 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:

```
53 # Bug: Accessing non-existing key
54 def get_value():
55     data = {"a": 1, "b": 2}
56     # Proper handling when key might not exist
57     return data.get("c", "Key not found")
58
59 print(get_value())
```

Output:

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

Explanation:

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:

```
61 # Bug: Infinite loop
62 def loop_example():
63     i = 0
64     while i < 5:
65         print(i)
66         i += 1
67 loop_example()
68
```

Output:

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

Explanation: 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:

```
72  # Bug: Wrong unpacking
73  a, b, _ = (1, 2, 3)
74  print(a, b)
75
```

Output:

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

Explanation: 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:

```
77 # Bug: Mixed indentation
78 def func():
79     x = 5
80     y = 10
81     return x + y
82 print(func())
```

Output:

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

Explanation: 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:

```
84 # Bug: Wrong import
85 import math
86 print(math.sqrt(16))
87
```

Output:

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

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

