

Ai Assisted Coding

Assignment-7.5

Name: A.Sai Shrehan

HT No:2303A2299

Batch:45

Task 1 (Mutable Default Argument – Function Bug)

Task: Analyze given code where a mutable default argument causes unexpected behavior. Use AI to fix it.

```
# Bug: Mutable default argument

def add_item(item, items=[]):
    items.append(item)
    return items

print(add_item(1))
print(add_item(2))
```

Expected Output: Corrected function avoids shared list bug.

The screenshot shows a code editor interface with a dark theme. At the top, there's a tab for 'assignment_7.5.py' and a status bar with icons for 'BLACKBOX', 'RUN', and '...'. Below the tabs, the code for 'add_item' is displayed:

```
assignment_7.5.py > add_item
1  #
2  def add_item(item, items=None):
3      if items is None:
4          items = []
5      else:
6          items = items.copy()
7      items.append(item)
8      return items
9 print(add_item(1))
10 print(add_item(2))
```

At the bottom, the terminal window shows the output of running the script:

```
PS C:\Users\Sushri\Documents\3-2\AI-Assisted coding> & C:/Users/Sushri/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/Sushri/Documents/3-2/AI-Assisted coding/assignment_7.5.py"
[1]
[2]
PS C:\Users\Sushri\Documents\3-2\AI-Assisted coding>
```

Task 2 (Floating-Point Precision Error)

Task: Analyze given code where floating-point comparison fails.

Use AI to correct with tolerance.

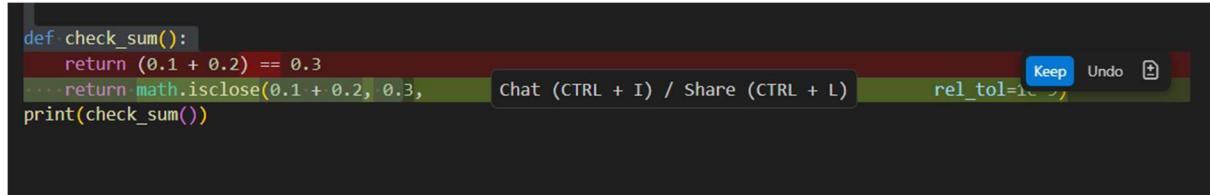
```
# Bug: Floating point precision issue

def check_sum():

    return (0.1 + 0.2) == 0.3

print(check_sum())
```

Expected Output: Corrected function



```
def check_sum():
    return (0.1 + 0.2) == 0.3
... return math.isclose(0.1 + 0.2, 0.3, rel_tol=1e-07)
print(check_sum())
```

Task 3 (Recursion Error – Missing Base Case)

Task: Analyze given code where recursion runs infinitely due to

missing base case. Use AI to fix.

```
# Bug: No base case
```

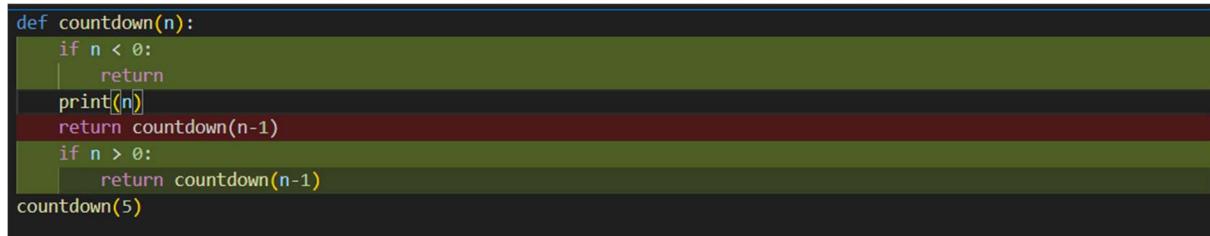
```
def countdown(n):

    print(n)

    return countdown(n-1)

countdown(5)
```

Expected Output : Correct recursion with stopping condition.



```
def countdown(n):
    if n < 0:
        return
    print(n)
    return countdown(n-1)
if n > 0:
    return countdown(n-1)
countdown(5)
```

Task 4 (Dictionary Key Error)

Task: Analyze given code where a missing dictionary key causes error. Use AI to fix it.

```
# Bug: Accessing non-existing key
```

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

Expected Output: Corrected with .get() or error handling.

A screenshot of a code editor window. The title bar says "fix missssing key bug in get_value function using .get function". The code area contains:

```
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")
print(get_value())
```

The line "return data.get("c", "Key not found")" is highlighted with a yellow background. A tooltip at the bottom right of the code area says "Chat (CTRL + I) / Share (CTRL + L)". The status bar at the bottom right shows "Auto".

Task 5 (Infinite Loop – Wrong Condition)

Task: Analyze given code where loop never ends. Use AI to detect and fix it.

Bug: Infinite loop

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
```

Expected Output: Corrected loop increments i.

A screenshot of a code editor window. The title bar says "avoid infinite running loop". The code area contains:

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1
loop_example()
```

The line "i += 1" is highlighted with a green background. A tooltip at the bottom right of the code area says "Chat (CTRL + I) / Share (CTRL + L)".

Task 6 (Unpacking Error – Wrong Variables)

Task: Analyze given code where tuple unpacking fails. Use AI to fix it.

```
# Bug: Wrong unpacking
```

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

Expected Output: Correct unpacking or using `_` for extra values`

```
a, b, c = 1, 2, 3
```

```
|
```

Task 7 (Mixed Indentation – Tabs vs Spaces)

Task: Analyze given code where mixed indentation breaks execution. Use AI to fix it.

```
# Bug: Mixed indentation
```

```
def func():
```

```
    x = 5
```

```
    y = 10
```

```
    return x+y
```

Expected Output : Consistent indentation applied.

Modify selected code

Add Context...

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

print(func()) Chat

Task: Analyze given code with incorrect import. Use AI to fix.

```
# Bug: Wrong import
```

```
import maths
```

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

Expected Output: Corrected to import math

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