

Lab Assignment-7.5

Name: M.Anjali

Hallticket:2303A51025

Batch:01

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.

Screenshots:

The screenshot shows a code editor window with the following code:

```
assg_07.py > add_item
  ⚡ | Add to chat (Alt+X) | Do not show again
  1  def add_item(item, items=[]):
      Fix the attached problems
      ⏎ × "items" is not defined × "item" is not defined × "items" is not defined × Code is structurally... × "item" is not accessed
      × "items" is not accessed
  2  items.append(item)
  3  return items
  4  print(add_item(1))
  5  print(add_item(2))
```

The code editor displays several ESLint-style errors in the status bar:

- × "items" is not defined
- × "item" is not defined
- × "items" is not defined
- × Code is structurally...
- × "item" is not accessed
- × "items" is not accessed

🐍 assg_07.py > ...

```
1  def add_item(item, items=[]):
2      items.append(item)
3      return items
4  print(add_item(1))
5  print(add_item(2))
```

🐍 assg_07.py > ⏷ add_item

Modify selected code ✓ ✕

∅ Add Context... Auto ▾

def add_item(item, items=[]):
 ↵ | Add to chat (Alt+X) | Do not show again

1 def add_item(item, items=None):
2 if items is None:
3 items = []
4 items.append(item)
5 return items
6 print(add_item(1))
7 print(add_item(2))

Keep Undo ⌛

🐍 assg_07.py > ...

```
1  def add_item(item, items=None):
2      if items is None:
3          items = []
4      items.append(item)
5      return items
6  print(add_item(1))
7  print(add_item(2))
```

output:

```
[1]  
[2]
```

Task 2 (Floating-Point Precision Error)

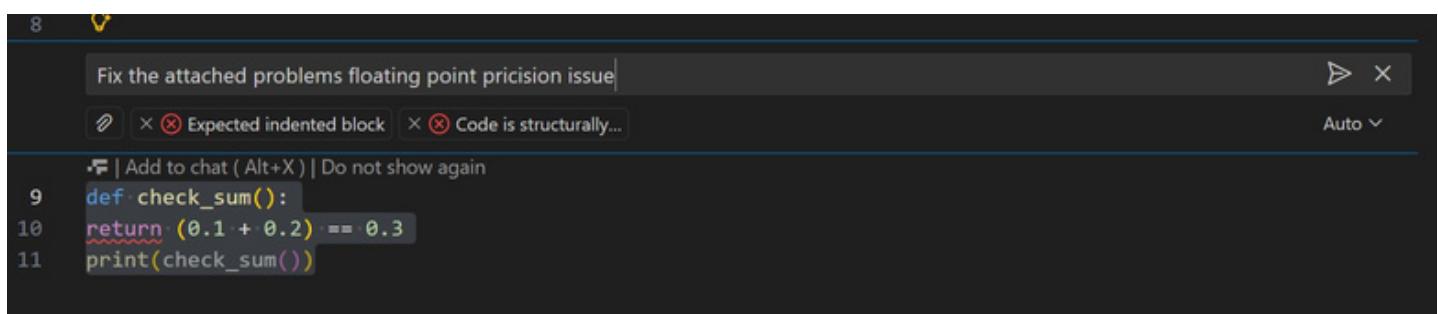
Task: Analyze given code where floating-point comparison fails. Use AI to correct with tolerance.

```
# Bug: Floating point precision issue def check_sum():
```

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

Expected Output: Corrected function

Screenshots:



```
8  Fix the attached problems floating point prcision issue  
9  ⚙ | Add to chat ( Alt+X ) | Do not show again  
10 def check_sum():  
11     return (0.1 + 0.2) == 0.3  
12     print(check_sum())
```



```
Keep Undo ⌂  
• | Add to chat ( Alt+X ) | Do not show again  
import math  
  
def check_sum():  
    return (0.1 + 0.2) == 0.3  
→     return math.isclose(0.1 + 0.2, 0.3)  
print(check_sum())
```

```
import math

def check_sum():
    return math.isclose(0.1 + 0.2, 0.3)
print(check_sum())
```

output:

```
True
```

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.

Screenshots:

correct the recursion with stopping condition

Add Context... Auto ▾

• Add to chat (Alt+X) | Do not show again

```
15 def countdown(n):
16     print(n)
17     return countdown(n-1)
18 countdown(5)
```

```
15 def countdown(n):
16     if n <= 0:
17         return
18     print(n)
19     return countdown(n-1)
20 countdown(5)
```

output:

```
5
4
3
2
1
```

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

Screenshots:

```
Modify selected code ✓ X
Add Context... Auto
• | Add to chat (Alt+X) | Do not show again
22 def get_value():
    data = {"a": 1, "b": 2}
    return data["c"]
23 - data = {"a": 1, "b": 2}
24 - return data.get("c", "Key not found")
25 print(get_value())
```

```
21
22     def get_value():
23         data = {"a": 1, "b": 2}
24         return data.get("c", "Key not found")
25     print(get_value())
```

output:

```
Key not found
```

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.

Screenshots:

```
27
28     def loop_example():
29         i = 0
30         while i < 5:
31             print(i)
→           i += 1
```

```
28
29     def loop_example():
30         i = 0
31         while i < 5:
32             print(i)
33             i += 1
34     loop_example()
```

output:

```
0  
1  
2  
3  
4
```

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.

Screenshots:

```
#wrong unpacking correct the code below code unpacking or using _ for extra values  
|  
| a, b, _ = (1, 2, 3)
```

```
36  
37     a, b, _ = (1, 2, 3)  
38     print(a, b)
```

output:

```
1 2
```

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():
```

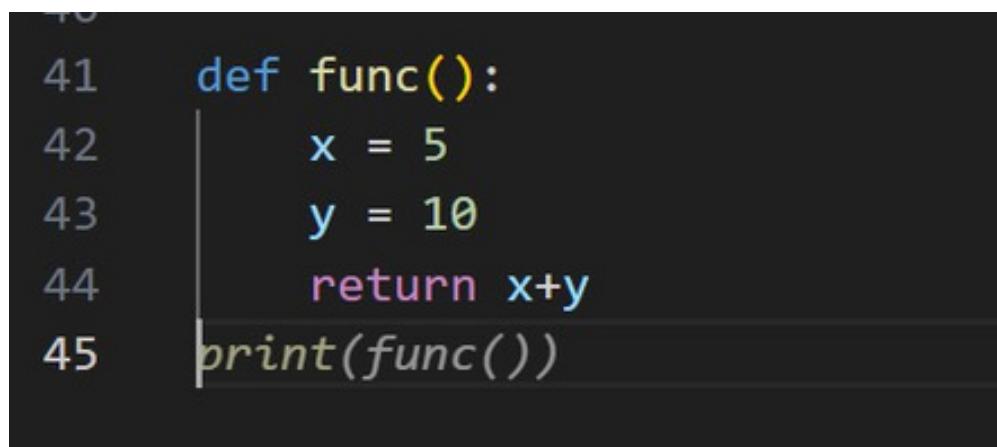
```
x = 5
```

```
y = 10
```

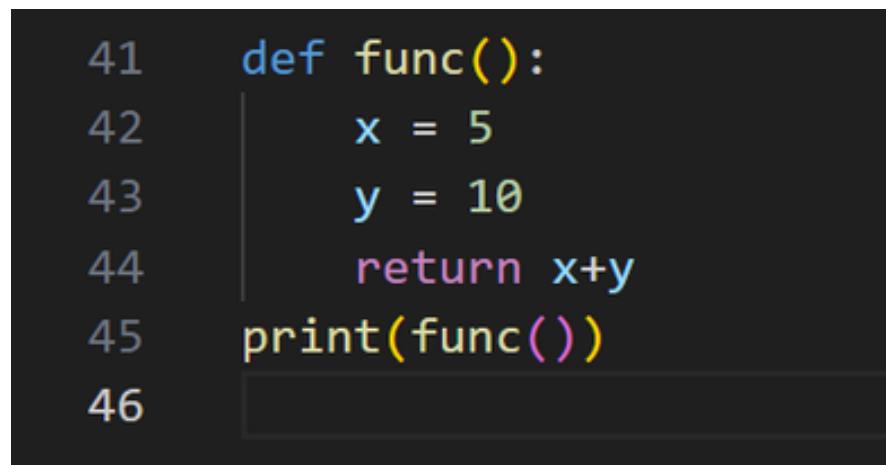
```
return x+y
```

Expected Output : Consistent indentation applied.

Screenshots:



```
18
41     def func():
42         x = 5
43         y = 10
44         return x+y
45     print(func())
```



```
18
41     def func():
42         x = 5
43         y = 10
44         return x+y
45     print(func())
46
```

output:

```
15
```

Task 8 (Import Error – Wrong Module Usage)

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

Screenshots:

A screenshot of a code editor window titled "correct the code". The code in the editor is:

```
46
47 import maths
48 print(maths.sqrt(16))
49
50
```

An error message at the top of the editor says: "Import "maths" could not be..." with a small icon next to it. There is also a tooltip-like message: "Add to chat (Alt+X) | Do not show again". The status bar at the bottom right shows "Auto".

A screenshot of a code editor window showing the corrected code. The code is:

```
46
47 import math
48 print(math.sqrt(16))
49
50
```

The word "maths" has been replaced by "math" in the corrected version. The status bar at the bottom right shows "Auto".

Output:



4.0

Task 9 (Unreachable Code – Return Inside Loop)

Task: Analyze given code where a return inside a loop prevents full iteration. Use AI to fix it.

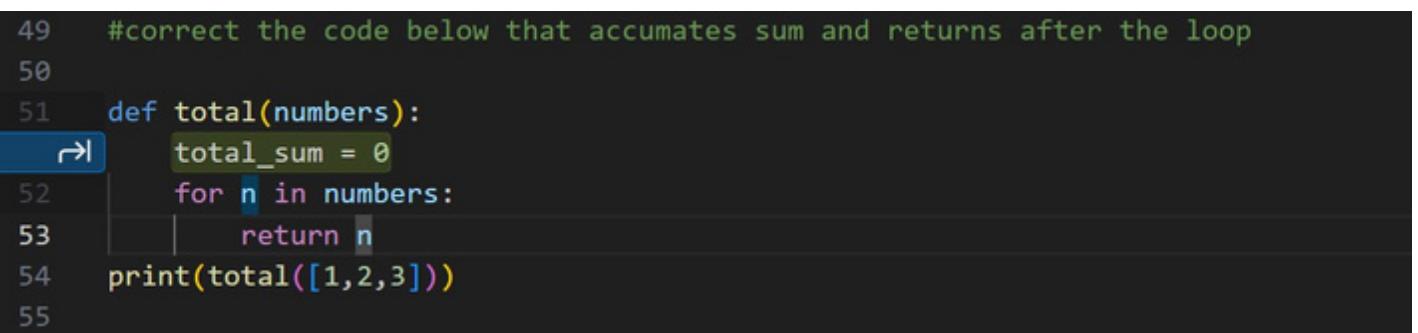
```
# Bug: Early return inside loop def total(numbers):
```

```
for n in numbers:
```

```
return n print(total([1,2,3]))
```

Expected Output: Corrected code accumulates sum and returns after loop.

Screenshots:



```
49 #correct the code below that accumates sum and returns after the loop
50
51 def total(numbers):
52     total_sum = 0
53     for n in numbers:
54         return n
55     print(total([1,2,3]))
```

```
49 #correct the code below that accumulates sum and returns after the loop
50
51 ✓ def total(numbers):
52     total_sum = 0
53 ✓     for n in numbers:
54         |     total_sum += n
55     return total_sum
56 print(total([1,2,3]))
57
58
```

output:

```
6
```

Task 10 (Name Error – Undefined Variable)

Task: Analyze given code where a variable is used before being defined. Let AI detect and fix the error.

```
# Bug: Using undefined variable def calculate_area():
```

```
return length * width print(calculate_area()) Requirements:
```

- Run the code to observe the error.
- Ask AI to identify the missing variable definition.
- Fix the bug by defining length and width as parameters.
- Add 3 assert test cases for correctness. Expected Output :
- Corrected code with parameters.
- AI explanation of the bug. Successful execution of assertions.

Screenshots:

58 |

→| *from turtle import width*

```
59     def calculate_area():
60         return length * width
61     print(calculate_area())
62
```

```
60     from turtle import width
61     def calculate_area(length, width):
62         return length * width
63     print(calculate_area(5, 10)) # Example values for length and width
64
```

output:

50

Task 11 (Type Error – Mixing Data Types Incorrectly)

Task: Analyze given code where integers and strings are added incorrectly. Let AI detect and fix the error.

```
# Bug: Adding integer and string def add_values():
```

```
return 5 + "10" print(add_values()) Requirements:
```

- Run the code to observe the error.
- AI should explain why int + str is invalid.
- Fix the code by type conversion (e.g., int("10") or str(5)).

- Verify with 3 assert cases. Expected Output #6:
- Corrected code with type handling.
- AI explanation of the fix.

Successful test validation.

```
66
67  def add_values():
68      return 5 + int("10") # The error occurs because we are trying to add an integer (5) and a string ("10")
69  print(add_values())
```

Screenshots:

output:

```
67  def add_values():
68 →|    return 5 + int("10")
69    print(add_values())
```

Task 12 (Type Error – String + List Concatenation)

Task: Analyze code where a string is incorrectly added to a list. # Bug: Adding string and list

```
def combine():
```

```
return "Numbers: " + [1, 2, 3] print(combine()) Requirements:
```

- Run the code to observe the error.
- Explain why str + list is invalid.
- Fix using conversion (str([1,2,3]) or " ".join()).
- Verify with 3 assert cases. Expected Output:

- Corrected code
- Explanation
- Successful test validation

Screenshots:

The screenshot shows a code editor window with a dark theme. At the top, there's a toolbar with a search bar labeled 'Modify selected code', a 'Keep (Ctrl+Enter)' button, and other icons. Below the toolbar, a message says 'Add to chat (Alt+X) | Do not show again'. The code area contains four lines of Python code:

```
71 def combine():
72     return "Numbers: " + str([1, 2, 3])
73
74 print(combine())
```

The line 'return "Numbers: " + str([1, 2, 3])' is highlighted with a red background, indicating an error. The line 'return "Numbers: " + ", ".join(map(str, [1, 2, 3]))' is highlighted with a green background, suggesting a fix. The word 'print' is highlighted in yellow.

The screenshot shows the same code editor window after the bug has been fixed. The code now looks like this:

```
70
71 def combine():
72     return "Numbers: " + ", ".join(map(str, [1, 2, 3]))
73 print(combine())
74
```

The entire code block is now displayed in a standard light gray background, indicating that the error has been resolved.

Output:

```
Numbers: 1, 2, 3
```

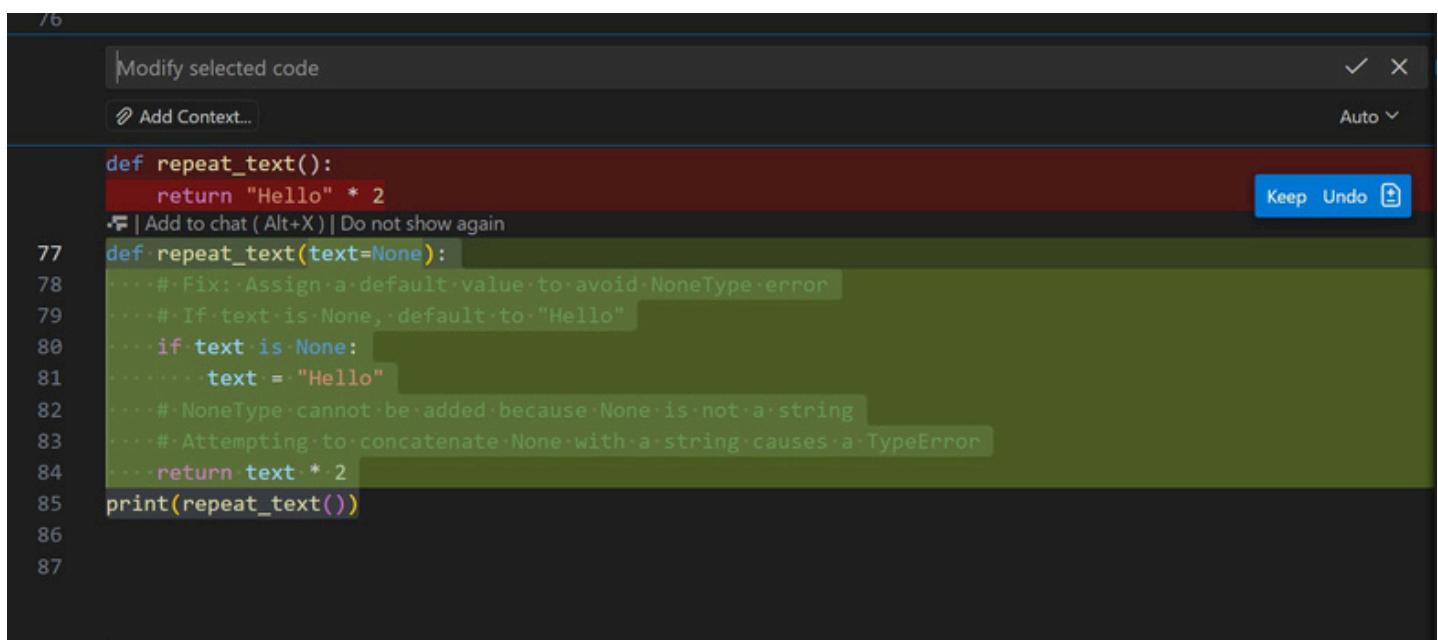
Task 13 (Type Error – Multiplying String by Float)

Task: Detect and fix code where a string is multiplied by a float. # Bug: Multiplying string by float

```
def repeat_text(): return "Hello" * 2.5 print(repeat_text()) Requirements:
```

- Observe the error.
- Explain why float multiplication is invalid for strings.
- Fix by converting float to int.
- Add 3 assert test cases.

Screenshots:



A screenshot of a code editor interface. At the top, there's a toolbar with a search bar labeled 'Modify selected code' and buttons for 'Keep', 'Undo', and a refresh icon. Below the toolbar, the code editor shows a file with the following content:

```
76
| Modify selected code
| Add Context...
Auto ▾
77
def repeat_text():
    return "Hello" * 2
| Add to chat (Alt+X) | Do not show again
78
def repeat_text(text=None):
    """# Fix: Assign a default value to avoid NoneType error
    # If text is None, default to "Hello"
    if text is None:
        text = "Hello"
    # NoneType cannot be added because None is not a string
    # Attempting to concatenate None with a string causes a TypeError
    return text * 2
85
print(repeat_text())
86
87
```

The line 'return "Hello" * 2' is highlighted with a red background, indicating an error. A tooltip above this line says '| Add to chat (Alt+X) | Do not show again'. The line 'if text is None:' is also highlighted with a red background, suggesting a fix or related code.

```
77 def repeat_text(text=None):
78     # Fix: Assign a default value to avoid NoneType error
79     # If text is None, default to "Hello"
80     if text is None:
81         text = "Hello"
82     # NoneType cannot be added because None is not a string
83     # Attempting to concatenate None with a string causes a TypeError
84     return text * 2
85 print(repeat_text())
86
87
```

output:

```
HelloHello
```

Task 15 (Type Error – Input Treated as String Instead of

Number)

Task: Fix code where user input is not converted properly. # Bug: Input remains string

```
def sum_two_numbers():
```

```
a = input("Enter first number: ")
```

```
b = input("Enter second number: ") return a + b print(sum_two_numbers()) Requirements:
```

- Explain why input is always string.
- Fix using int() conversion.
- Verify with assert test cases.

Screenshot

87

Modify selected code

✓ ✕

 Add Context...

Auto

· | Add to chat (Alt+X) | Do not show again

```
88 def sum_two_numbers():
    a = input("Enter first number: ")
    b = input("Enter second number: ")
    # input() always returns a string, even if the user enters numbers
    # Attempting to add two strings will concatenate them instead of summing
    # Example: "5" + "3" = "53" (string concatenation, not addition)
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    # Convert strings to integers using int() before performing arithmetic
    return a + b
96
97 print(sum_two_numbers())
98
99
```

87

```
88 def sum_two_numbers():
    # input() always returns a string, even if the user enters numbers
    # Attempting to add two strings will concatenate them instead of summing
    # Example: "5" + "3" = "53" (string concatenation, not addition)
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    # Convert strings to integers using int() before performing arithmetic
    return a + b
96
97 print(sum_two_numbers())
98
```

output:

```
Enter first number: 12
Enter second number: 19
31
```