

# Lab – 7.5

## Course : AI Assisted Coding

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

---

Student Details:

Name : Mohammed Sabir

Hall Ticket No : 2303A51506

Batch 22

---

### Task-01

Final Optimal Prompt

---

Fix only the errors in the voting eligibility code and add validation for invalid or non-numeric age input without changing the main logic

Code Screenshot

---

The screenshot shows a code editor window titled "Lab-7-Task-01/05". The code is as follows:

```
1 # ---Lab-7-Task-01/05--- Chat Ctrl+L Edit Ctrl+I
2 fix the
3 # Add Context...
4
5 # Bug: Mutable default argument
6 def add_item(item, items=[]):
7     items.append(item)
8     return items
9
10 # Bug: Mutable default argument - FIXED
11 def add_item(item, items=None):
12     if items is None:
13         items = []
14     items.append(item)
15     return items
16
17 print(add_item(1))
18 print(add_item(2))
```

The code editor highlights the first two lines of the original code in red, indicating they are incorrect. The corrected code from line 11 onwards is highlighted in green, indicating it is the final optimal solution.

Output Screenshot

---

The screenshot shows a terminal window with the following text:

```
PS D:\6th Sem 2025-26\AI-Assisted Coding> & "D:\python26/AI-Assisted Coding/Assignment-07(Lab-7.5).py"
[1]
[2]
PS D:\6th Sem 2025-26\AI-Assisted Coding>
```

---

Explanation/Justification/Observation (100 words / 5 – 6 sentence)

The bug occurs because the default list is shared across all function calls. AI helps identify the issue and fixes it by replacing the mutable default with None. A new list is created on each call, preventing unexpected behaviour.

## Task-02

---

Final Optimal Prompt

Correct the string vowel/consonant code and add error handling for numbers, symbols, or empty input while keeping the original structure.

---

Code Screenshot

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

```
# Bug: Floating point precision issue
def check_sum():
    return (0.1 + 0.2) == 0.3
# AI suggestion: Using abs() to handle floating-point precision
# return abs((0.1 + 0.2) - 0.3) < 1e-9
print(check_sum())
```

---

Output Screenshot

The screenshot shows a terminal window with the following text:

```
PS D:\6th Sem 2025-26\AI-Assisted Coding\Assignment-07> & "D:\python26/AI-Assisted Coding/Assignment-07(Lab-7.5).py"
True
PS D:\6th Sem 2025-26\AI-Assisted Coding\Assignment-07>
```

---

Explanation/Justification/Observation (100 words / 5 – 6 sentence)

Floating-point arithmetic cannot reliably compare decimal values directly. AI suggests using a tolerance check ( $\text{abs}(\text{diff}) < \text{epsilon}$ ) to avoid precision errors. This ensures a correct and reliable comparison.

## Task-03

### Final Optimal Prompt

---

Fix errors in the class-based library program and add simple input validation without rewriting or optimizing the logic

### Code Screenshot

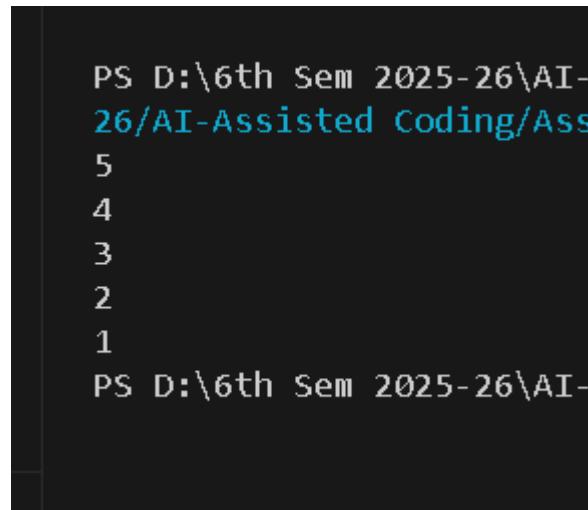
---



```
# Bug: No base case
def countdown(n):
    if not isinstance(n, int) or n < 0:
        return None
    if n == 0:
        return
    print(n)
    return countdown(n-1)
countdown(5)
```

### Output Screenshot

---



```
PS D:\6th Sem 2025-26\AI-
26/AI-Assisted Coding/Ass
5
4
3
2
1
PS D:\6th Sem 2025-26\AI-
```

### Explanation/Justification/Observation (100 words / 5 – 6 sentence)

---

The original function lacked a base case, causing infinite recursion. AI introduces a stopping condition ( $n \leq 0$ ), allowing the function to terminate safely.

## Task-04

### Final Optimal Prompt

Correct the attendance class code and add basic validation for wrong or empty inputs without altering the program design

### Code Screenshot

The screenshot shows a code editor window titled "Modify selected code". The code is as follows:

```
# Bug: Accessing non-existing key
def get_value():
    def get_value(key="c"):
        data = {"a": 1, "b": 2}
        return data[key]
    if key in data:
        return data[key]
    else:
        return None
print(get_value())
```

### Output Screenshot

The screenshot shows a terminal window with the following output:

```
26/AI-Assisted Coding/Assignment-07(Lab-7.5).py"
Enter a key (a, b, or c): c
None
PS D:\6th Sem 2025-26\AI-Assisted Coding> & "D:\py
26/AI-Assisted Coding/Assignment-07(Lab-7.5).py"
Enter a key (a, b, or c): a
1
PS D:\6th Sem 2025-26\AI-Assisted Coding> & "D:\py
26/AI-Assisted Coding/Assignment-07(Lab-7.5).py"
Enter a key (a, b, or c): b
2
```

### Explanation/Justification/Observation (100 words / 5 – 6 sentence)

Accessing a missing key raises a Key Error. AI fixes this using .get() with a default message, preventing crashes and improving program reliability.

## Tack-05

### Final Optimal Prompt

Fix the ATM menu errors and add checks for invalid menu choices or non-numeric inputs while keeping the same program flow

### Code Screenshot

The screenshot shows a code editor window titled "Lab - Task 05, 05". The code is as follows:

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

The code is highlighted with syntax coloring. A tooltip "Chat Ctrl+L Edit Ctrl+I" is visible above the code area. The line "i += 1" is selected.

### Output Screenshot

The screenshot shows a terminal window with the following output:

```
PS D:\6th Sem 2025-26\AI-Assisted Coding> & "D:\p
26/AI-Assisted Coding\Assignment-07(Lab-7.5).py"
0
1
2
3
4
PS D:\6th Sem 2025-26\AI-Assisted Coding>
```

### Explanation/Justification/Observation (100 words / 5 – 6 sentence)

The loop condition was correct, but the variable never changed, causing an infinite loop. AI fixes it by incrementing I inside the loop.

## Tack-06

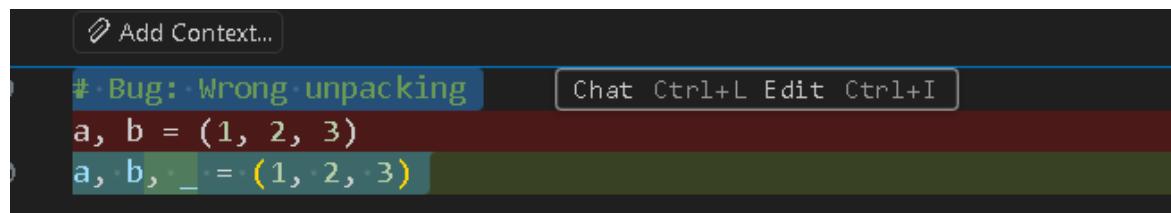
### Final Optimal Prompt

---

Correct the tuple unpacking issue without altering the original values

### Code Screenshot

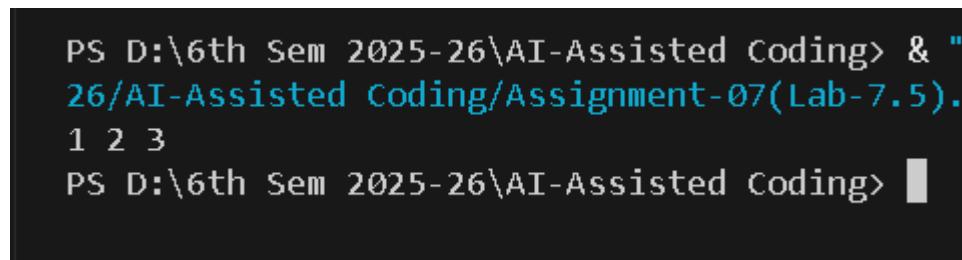
---



The screenshot shows a code editor interface. At the top, there is a button labeled 'Add Context...'. Below it, a status bar displays the text '# Bug: Wrong unpacking' on the left and 'Chat Ctrl+L Edit Ctrl+I' on the right. The main code area contains two lines of Python code: 'a, b = (1, 2, 3)' and 'a, b, \_ = (1, 2, 3)'. The first line is in a standard font, while the second line has the variable '\_' highlighted in green, indicating it is being added by AI to handle the extra value from the tuple.

### Output Screenshot

---



The screenshot shows a terminal window with a black background and white text. It displays the command 'PS D:\6th Sem 2025-26\AI-Assisted Coding> & "26/AI-Assisted Coding/Assignment-07(Lab-7.5).py"', followed by the output '1 2 3'. The terminal prompt 'PS D:\6th Sem 2025-26\AI-Assisted Coding>' is visible at the bottom.

### Explanation/Justification/Observation (100 words / 5 – 6 sentence)

---

The bug occurs because the tuple contains 3 values but only 2 variables. AI corrects the mismatch by adding another variable or using `_` to ignore the extra value.

## Tack-07

### Final Optimal Prompt

---

Fix the mixed indentation so the code runs properly; do not change any logic

### Code Screenshot

---

The screenshot shows a code editor window titled '#---Lab-7-Task-07/08 ---'. The code contains several lines of Python, with some lines having inconsistent indentation. The lines are color-coded: lines 2, 3, 4, 5, 6, and 7 are in different colors (blue, green, yellow, red, orange, and purple), while line 8 is in grey. The code is as follows:

```
2 #---Lab-7-Task-07/08 ---
3 # Bug: Mixed indentation
4 def func():
5     x = 5
6     y = 10
7     y = 10
8     return x+y
```

The status bar at the bottom right shows 'Output'.

### Screenshot

---

The screenshot shows a terminal window with the following text:

```
PS D:\6th Sem 2025-26\AI-Assisted Coding> & "D:\python 3.13.6\python.exe" "d:/6
26/AI-Assisted Coding/Assignment-07(Lab-7.5).py"
15
PS D:\6th Sem 2025-26\AI-Assisted Coding>
```

### Explanation/Justification/Observation (100 words / 5 – 6 sentence)

---

Tabs and spaces were mixed, causing an indentation error. AI normalizes indentation using consistent spaces, ensuring the code runs correctly.

## Tack-08

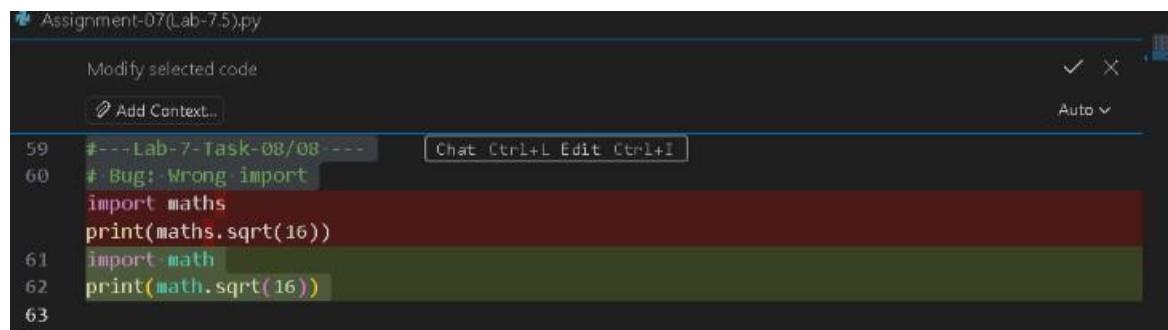
### Final Optimal Prompt

---

Fix only the module import error so the sqrt function works correctly

### Code Screenshot

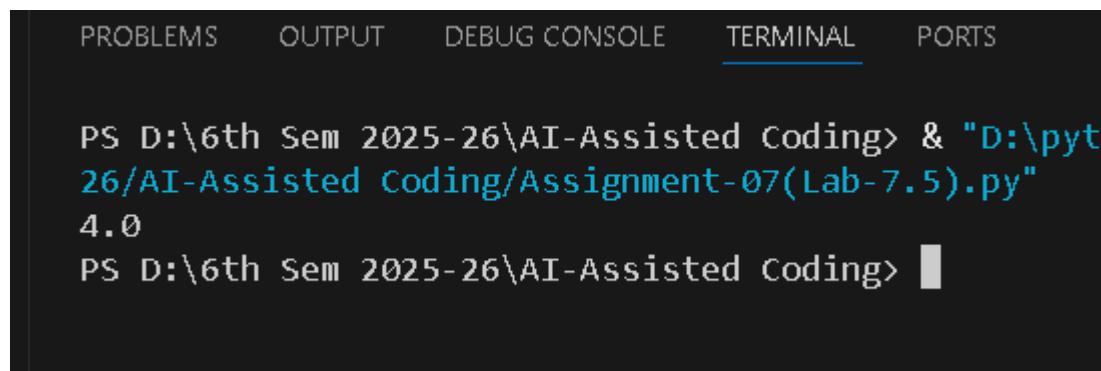
---



```
#--Lab-7-Task-08/08--- Chat Ctrl+L Edit Ctrl+I
# Bug: Wrong import
import maths
print(maths.sqrt(16))
import math
print(math.sqrt(16))
```

### Output Screenshot

---



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\6th Sem 2025-26\AI-Assisted Coding> & "D:\pyth
26/AI-Assisted Coding\Assignment-07(Lab-7.5).py"
4.0
PS D:\6th Sem 2025-26\AI-Assisted Coding>
```

### Explanation/Justification/Observation (100 words / 5 – 6 sentence)

---

The module name was incorrect (maths does not exist). AI corrects it to math, ensuring proper module import and execution.