

## ASSIGNMENT-7.5

M.AKHIL REDDY

2303A52315

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 list `items` is **shared across all function calls**
- That's why the second call keeps data from the first call

```
7.5.py x
7.5.py > ...
1 def add_item(item, items=None):
2     if items is None:
3         items = []
4         items.append(item)
5     return items
6
7 print(add_item(1))
8 print(add_item(2))
9
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ive/Desktop/akhils ai coding/7.5.py"
[1]
[2]
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```

## Task 2 (Floating-Point Precision Error)

Task2: Analyze given code where floating-point comparison fails. Use AI to correct with tolerance.

# Bug: Floating point precision issue

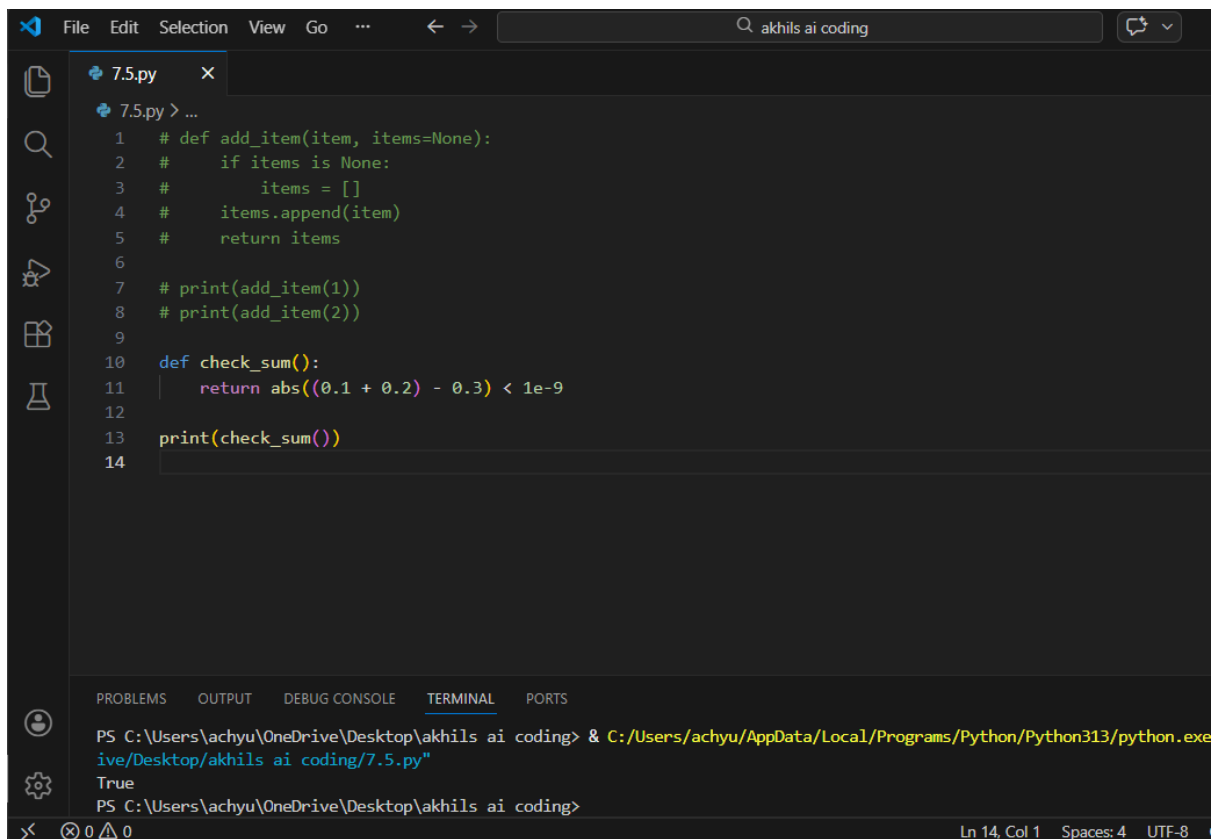
```
def check_sum():
```

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

```
print(check_sum())
```

Expected Output: Corrected function

Floating-point numbers are stored in **binary**, so some decimal values cannot be represented exactly.



```
File Edit Selection View Go ... ← → 🔍 akhils ai coding
```

```
7.5.py x
```

```
7.5.py > ...
```

```
1 # def add_item(item, items=None):
2 #     if items is None:
3 #         items = []
4 #     items.append(item)
5 #     return items
6
7 # print(add_item(1))
8 # print(add_item(2))
9
10 def check_sum():
11     return abs((0.1 + 0.2) - 0.3) < 1e-9
12
13 print(check_sum())
14
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe
ive/Desktop/akhils ai coding/7.5.py"
True
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```

```
< 0 0 0 Ln 14, Col 1 Spaces: 4 UTF-8
```

### 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)

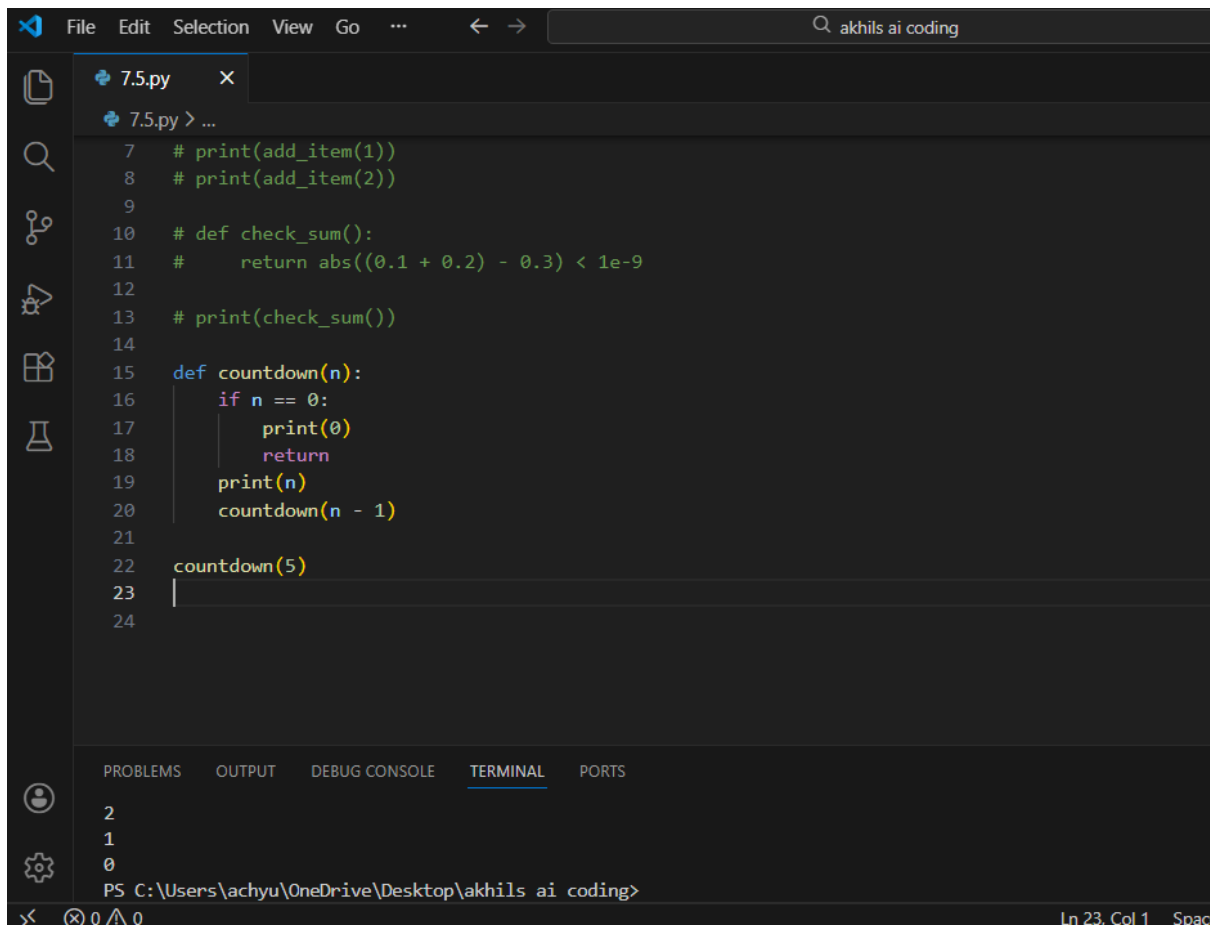
countdown(5)
```

Expected Output : Correct recursion with stopping condition.

There is **no base case**, so the function keeps calling itself forever.

This leads to:

- Infinite recursion
- `RecursionError: maximum recursion depth exceeded`



```
File Edit Selection View Go ... ← → akhils ai coding
7.5.py x
7.5.py > ...
7 # print(add_item(1))
8 # print(add_item(2))
9
10 # def check_sum():
11 #     return abs((0.1 + 0.2) - 0.3) < 1e-9
12
13 # print(check_sum())
14
15 def countdown(n):
16     if n == 0:
17         print(0)
18         return
19     print(n)
20     countdown(n - 1)
21
22 countdown(5)
23
24
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
2
1
0
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```

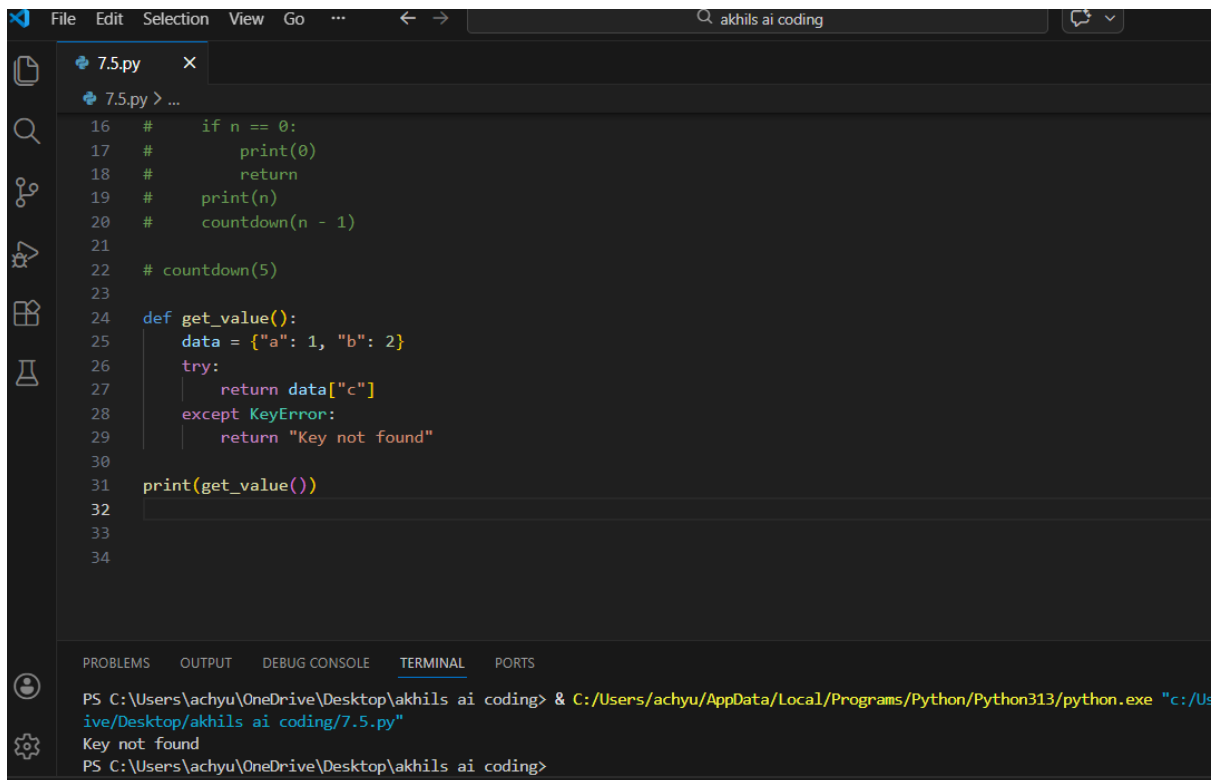
#### Task 4 (Dictionary Key Error)

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.



The screenshot shows a code editor with a file named `7.5.py`. The code includes a `countdown` function and a `get_value` function. The `get_value` function attempts to access a key `"c"` in a dictionary, which results in a `KeyError`. The terminal at the bottom shows the command to run the script and the resulting output: `Key not found`.

```
16 # if n == 0:
17 #     print(0)
18 #     return
19 #     print(n)
20 #     countdown(n - 1)
21
22 # countdown(5)
23
24 def get_value():
25     data = {"a": 1, "b": 2}
26     try:
27         return data["c"]
28     except KeyError:
29         return "Key not found"
30
31 print(get_value())
32
33
34
```

Terminal Output:

```
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"
Key not found
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```

given very well

#### Task 5 (Infinite Loop – Wrong Condition)

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

# Bug: Infinite loop

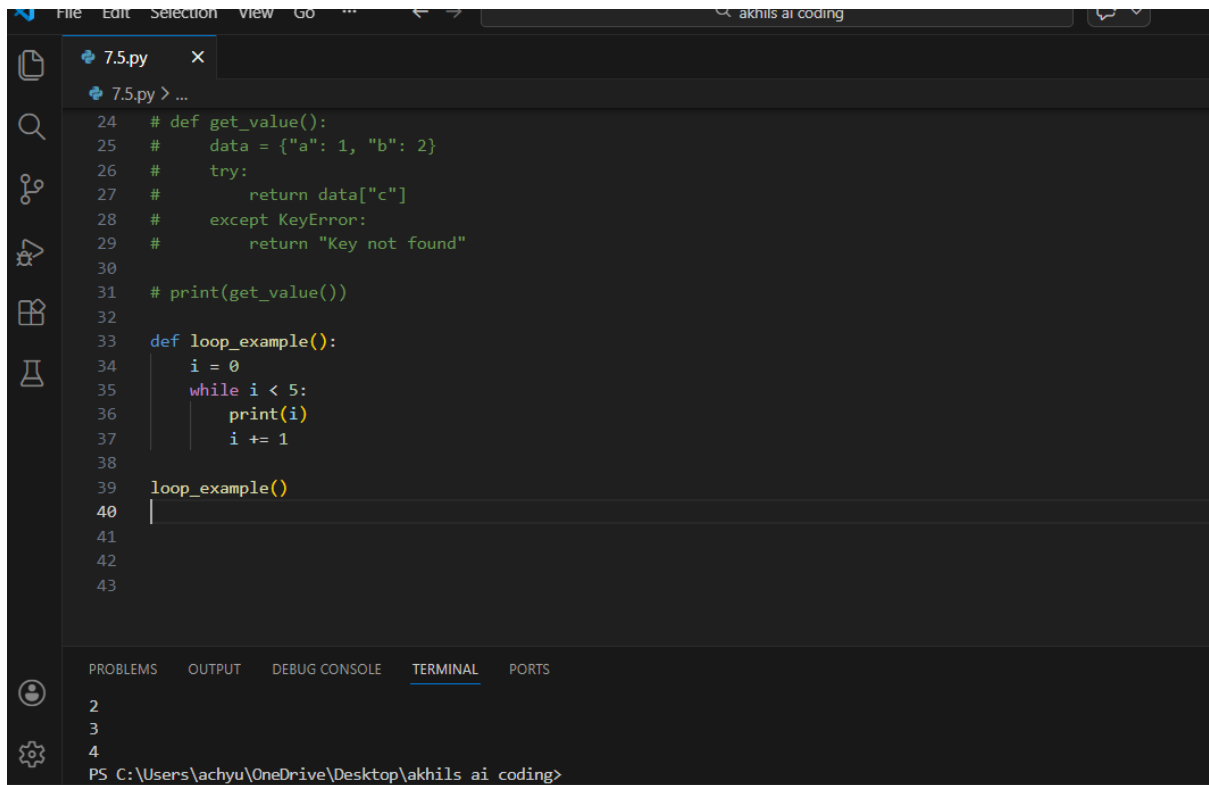
```
def loop_example():
```

```
    i = 0
```

```
    while i < 5:
```

```
        print(i)
```

Expected Output: Corrected loop increments i.



```
24 # def get_value():
25 #     data = {"a": 1, "b": 2}
26 #     try:
27 #         return data["c"]
28 #     except KeyError:
29 #         return "Key not found"
30
31 # print(get_value())
32
33 def loop_example():
34     i = 0
35     while i < 5:
36         print(i)
37         i += 1
38
39 loop_example()
40
41
42
43
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
2
3
4
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```

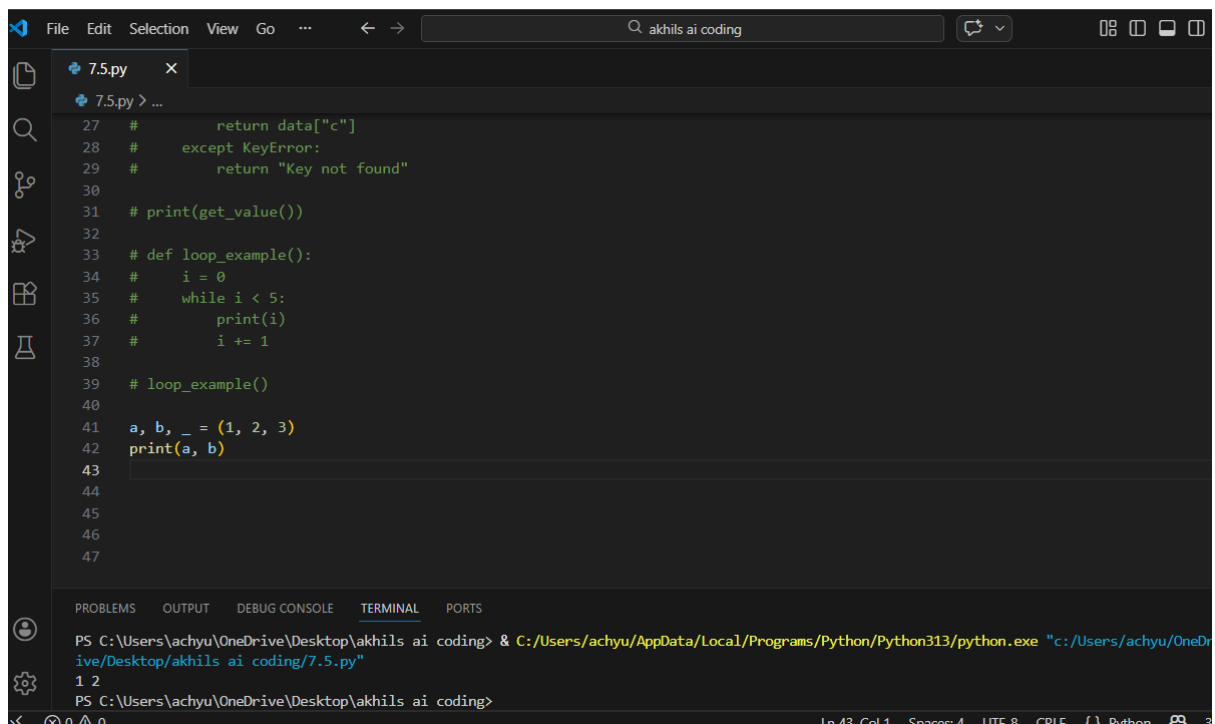
### Task 6 (Unpacking Error – Wrong Variables)

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.



```
File Edit Selection View Go ... ← → 🔍 akhils ai coding
7.5.py x
7.5.py > ...
27 #         return data["c"]
28 #     except KeyError:
29 #         return "Key not found"
30
31 # print(get_value())
32
33 # def loop_example():
34 #     i = 0
35 #     while i < 5:
36 #         print(i)
37 #         i += 1
38
39 # loop_example()
40
41 a, b, _ = (1, 2, 3)
42 print(a, b)
43
44
45
46
47
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"
1 2
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```

### 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.

```
File Edit Selection View Go ... akhils ai coding
7.5.py
34 # i = 0
35 # while i < 5:
36 #     print(i)
37 #     i += 1
38
39 # loop_example()
40
41 # a, b, _ = (1, 2, 3)
42 # print(a, b)
43
44 def func():
45     x = 5
46     y = 10
47     return x + y
48
49 print(func())
50
51
52
53
54
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"
15
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```

## 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

```
File Edit Selection View Go ... akhils ai coding
7.5.py
37 # i += 1
38
39 # loop_example()
40
41 # a, b, _ = (1, 2, 3)
42 # print(a, b)
43
44 # def func():
45 #     x = 5
46 #     y = 10
47 #     return x + y
48
49 # print(func())
50
51 import math
52 print(math.sqrt(16))
53
54
55
56
57
58
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"
4.0
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```



### 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

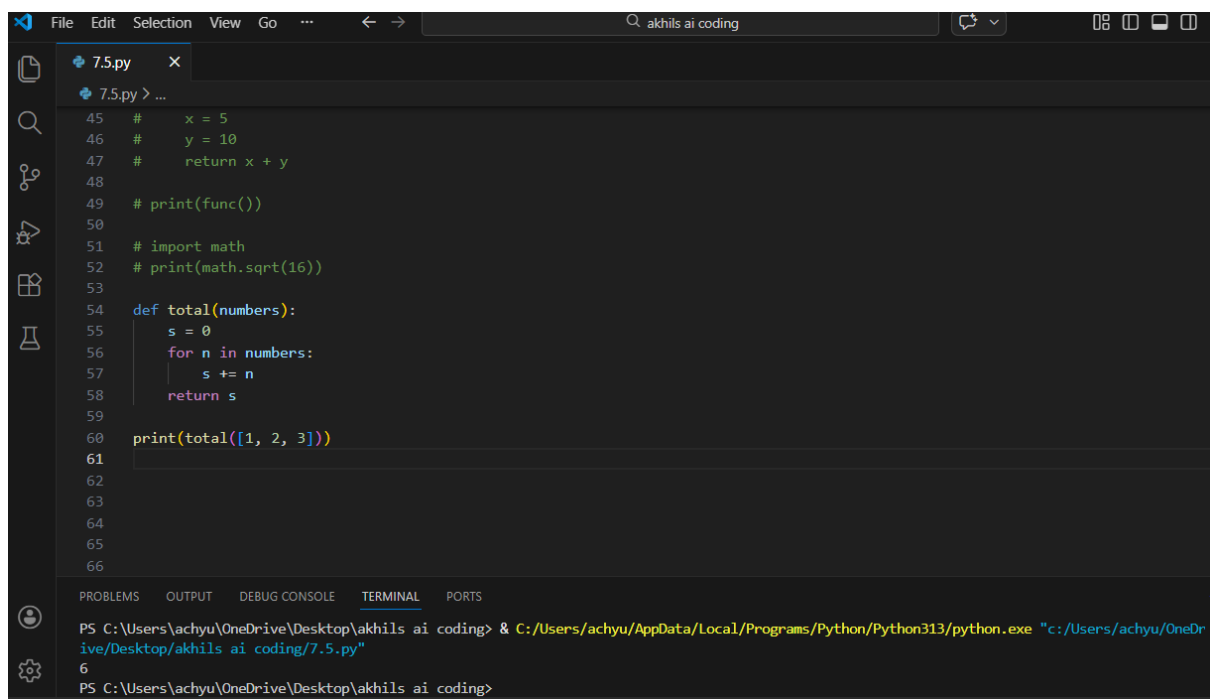
```
def total(numbers):
```

```
    for n in numbers:
```

```
        return n
```

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

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



The screenshot shows a code editor with a file named 7.5.py. The code contains several comments and a function definition. The function `total` is defined with a loop that returns the first element `n` immediately, which is the bug. Below the function, `print(total([1, 2, 3]))` is called. The terminal at the bottom shows the command to run the file, which outputs the number 6, indicating that the sum of 1, 2, and 3 was calculated despite the early return in the function.

```
45 # x = 5
46 # y = 10
47 # return x + y
48
49 # print(func())
50
51 # import math
52 # print(math.sqrt(16))
53
54 def total(numbers):
55     s = 0
56     for n in numbers:
57         s += n
58         return s
59
60 print(total([1, 2, 3]))
61
62
63
64
65
66
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"

6

PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>

### 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.

```

49 # print(func())
50
51 # import math
52 # print(math.sqrt(16))
53
54 # def total(numbers):
55 #     s = 0
56 #     for n in numbers:
57 #         s += n
58 #     return s
59
60 # print(total([1, 2, 3]))
61
62 def calculate_area(length, width):
63     return length * width
64
65 assert calculate_area(5, 4) == 20
66 assert calculate_area(10, 2) == 20
67 assert calculate_area(3, 3) == 9
68
69 print("All test cases passed")
70

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"
All test cases passed
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>

```

## 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.

The screenshot shows a VS Code editor window with a file named `7.5.py`. The code contains several test cases and a function `add_values()`. The terminal window at the bottom shows the command to run the file and the output "All test cases passed".

```

63 #     return length * width
64
65 # assert calculate_area(5, 4) == 20
66 # assert calculate_area(10, 2) == 20
67 # assert calculate_area(3, 3) == 9
68
69 # print("All test cases passed")
70
71 def add_values():
72     return 5 + int("10")
73
74 assert add_values() == 15
75 assert 3 + int("7") == 10
76 assert int("4") + 6 == 10
77
78 print("All test cases passed")
79
80
81
82
83
84

```

Terminal Output:

```

PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"
All test cases passed
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>

```

## 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

```

66 # assert calculate_area(10, 2) == 20
67 # assert calculate_area(3, 3) == 9
68
69 # print("All test cases passed")
70
71 # def add_values():
72 #     return 5 + int("10")
73
74 # assert add_values() == 15
75 # assert 3 + int("7") == 10
76 # assert int("4") + 6 == 10
77
78 # print("All test cases passed")
79
80 def combine():
81     return "Numbers: " + str([1, 2, 3])
82
83 assert combine() == "Numbers: [1, 2, 3]"
84 assert "A" + str([1]) == "A[1]"
85 assert "List: " + str([]) == "List: []"
86
87 print("All test cases passed")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"
All test cases passed
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>

```

### 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.

```

81 #     return "Numbers: " + str([1, 2, 3])
82
83 # assert combine() == "Numbers: [1, 2, 3]"
84 # assert "A" + str([1]) == "A[1]"
85 # assert "List: " + str([]) == "List: []"
86
87 # print("All test cases passed")
88
89 def repeat_text():
90     return "Hello" * int(2.5)
91
92 assert repeat_text() == "HelloHello"
93 assert "A" * int(3.9) == "AAA"
94 assert "Hi" * int(1.1) == "Hi"
95
96 print("All test cases passed")
97
98
99
100
101
102

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```

PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/Desktop/akhils ai coding/7.5.py"
All test cases passed
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>

```

## Task 14 (Type Error – Adding None to Integer)

Task: Analyze code where None is added to an integer.

# Bug: Adding None and integer

```
def compute():
```

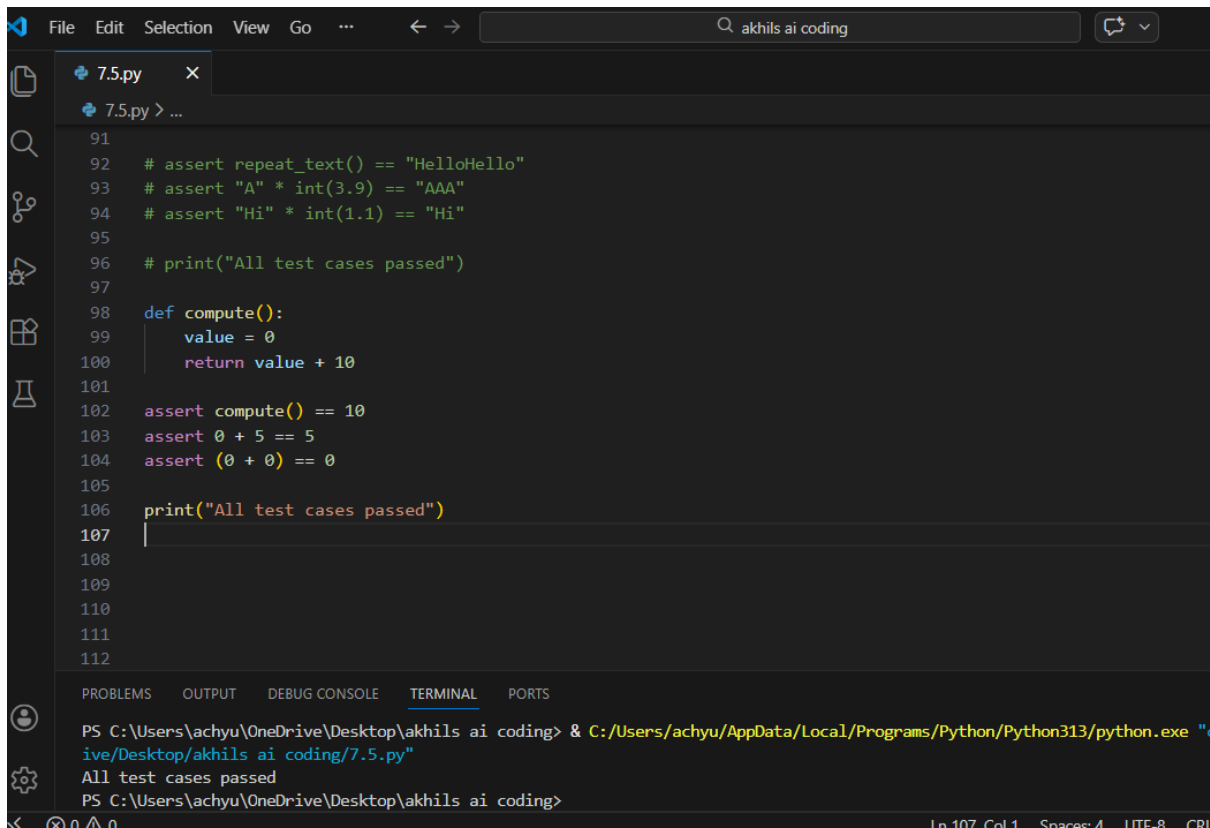
```
    value = None
```

```
    return value + 10
```

```
print(compute())
```

Requirements:

- Run and identify the error.
- Explain why NoneType cannot be added.
- Fix by assigning a default value.
- Validate using asserts.



```
91
92 # assert repeat_text() == "HelloHello"
93 # assert "A" * int(3.9) == "AAA"
94 # assert "Hi" * int(1.1) == "Hi"
95
96 # print("All test cases passed")
97
98 def compute():
99     value = 0
100     return value + 10
101
102 assert compute() == 10
103 assert 0 + 5 == 5
104 assert (0 + 0) == 0
105
106 print("All test cases passed")
107
108
109
110
111
112
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "C:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"

All test cases passed

PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>

## 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: ")
```

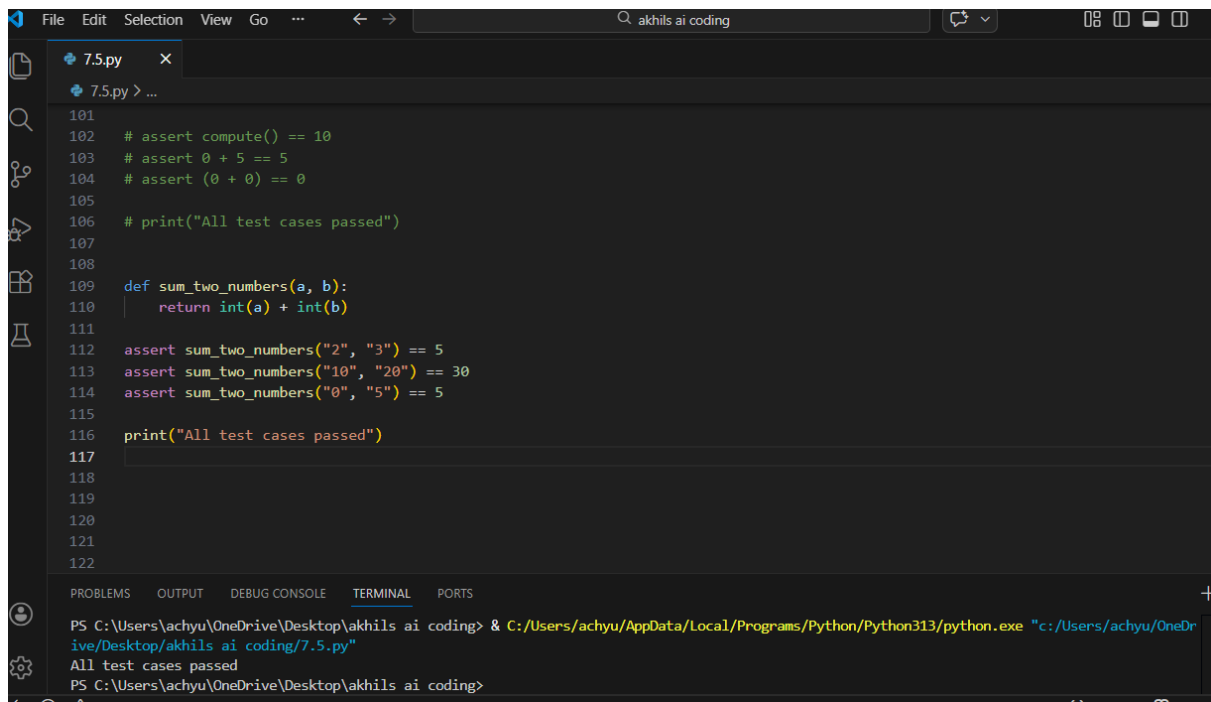
```
    b = input("Enter second number: ")
```

```
    return a + b
```

```
print(sum_two_numbers())
```

Requirements:

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



The image shows a Visual Studio Code editor window with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, and a search bar containing 'akhils ai coding'. The left sidebar shows icons for Explorer, Search, Source Control, Run and Debug, Extensions, and Testing. The main editor area displays a Python file named '7.5.py' with the following code:

```
101
102 # assert compute() == 10
103 # assert 0 + 5 == 5
104 # assert (0 + 0) == 0
105
106 # print("All test cases passed")
107
108
109 def sum_two_numbers(a, b):
110     return int(a) + int(b)
111
112 assert sum_two_numbers("2", "3") == 5
113 assert sum_two_numbers("10", "20") == 30
114 assert sum_two_numbers("0", "5") == 5
115
116 print("All test cases passed")
117
118
119
120
121
122
```

Below the editor, the 'TERMINAL' tab is active, showing the command used to run the script and its output:

```
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding> & C:/Users/achyu/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/achyu/OneDrive/Desktop/akhils ai coding/7.5.py"
All test cases passed
PS C:\Users\achyu\OneDrive\Desktop\akhils ai coding>
```