



## School of Computer Science and Artificial Intelligence

### Lab Assignment-7.5

**Course Title : AI Assistant Coding**  
**Name of Student : Shaik Naved Ahmed**  
**Enrollment No. : 2303A54053**  
**Batch No. : 47-B**

#### **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.**

#### **EXPLANATION:**

The original code uses a mutable default argument (a list) which is shared across all calls to the function. This means that when you append an item to the list in one call, it affects all subsequent calls that use the default list. In the fixed code, we use None as the default value and create a new list inside the function if items is None. This ensures that each call to add\_item gets its own separate list.

#### **SCREENSHOT OF GENERATED CODE:**

```
1 #Task 1
2 #Buggy Code:
3 '''def add_item(item, items=[]):
4     [items.append(item)
5      ]|return items
6     print(add_item1())
7     print(add_item2())'''
8 #Fixed Code:
9 def add_item(item, items=None):
10     if items is None:
11         items = []
12     items.append(item)
13     return items
14 print(add_item1())
15 print(add_item2())
16 #Explanation: The original code uses a mutable default argument (a list) which is shared across all calls to the function. This means that when you append an item to the list in one call, it affects all subsequent calls that use the default list. In the fixed code, we use None as the default value and create a new list inside the function if items is None. This ensures that each call to add_item gets its own separate list.
```

(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py

[1]  
[2]

(base) navedahmedshaik@Naveds-MacBook-Air AIAC %

## 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**

## EXPLANATION:

The original code checks if the sum of 0.1 and 0.2 is exactly equal to 0.3, which can lead to a false result due to floating-point precision issues. In the fixed code, we check if the absolute difference between (0.1 + 0.2) and 0.3 is smaller than a very small number (1e-9), which accounts for the precision error and gives a more accurate result.

## SCREENSHOT OF GENERATED CODE:

```
lab-7.1.py > ...
1 #Task 1
2 #Buggy Code:
3 """def add_item(item, items=[]):
4     [items.append(item)
5      ] [return items
6      ]
7 print(add_item())
8 print(add_item(2))
9 #Fixed Code:
10 def add_item(item, items=None):
11     if items is None:
12         items = []
13     items.append(item)
14     return items
15 print(add_item(1))
16 print(add_item(2))
17 #Explanation: The original code uses a mutable default argument (a list) which is shared across all calls to the function. This means that when you append an item
18 #Task 2
19 #Buggy Code:
20 """def check_sum():
21     return (0.1 + 0.2) == 0.3
22 print(check_sum())
23 #Fixed Code:
24 def check_sum():
25     return abs(0.1 + 0.2 - 0.3) < 1e-9
26 print(check_sum())
27 #Explanation: The original code checks if the sum of 0.1 and 0.2 is exactly equal to 0.3, which can lead to a false result due to floating-point precision issues.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● [base] navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py
[1]
[2]
● [base] navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py
[1]
[2]
True
● [base] navedahmedshaik@Naveds-MacBook-Air AIAC % 
```

## 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.**

## EXPLANATION:

The original code does not have a base case to stop the recursion, which leads to a maximum recursion depth error when n becomes negative. In the fixed code, we add a base case that checks if n is less than 0, at which point the function returns and stops the recursion. This allows the countdown to work correctly without causing an error.

## SCREENSHOT OF GENERATED CODE:

The screenshot shows a VS Code interface with a dark theme. The Explorer sidebar on the left lists files under the 'AIAC' folder, including 'example.py', 'Lab-01.pages', 'Lab-01.pdf', 'Lab-02.docx', 'Lab-02.pages', 'Lab-02.pdf', 'Lab-3-1.pages', 'Lab-3-1.pdf', 'Lab-3-1.py', 'Lab-4-1.pages', 'Lab-4-1.pdf', 'Lab-4-1.py', 'Lab-4-3.pages', 'Lab-4-3.pdf', 'Lab-4-3.py', 'Lab-5-1-4053.pages', 'Lab-5-1-4053.pdf', 'Lab-5-1.py', 'Lab-6-3-4053.pdf', 'Lab-6-3-5053.pages', 'Lab-6-3.py', 'Lab-7-1.py', 'users\_secure.txt', and 'users.txt'. The 'lab-7-1.py' file is open in the editor tab, containing the following code:

```
29     #task 3
30     #Buggy Code:
31     '''def countdown(n):
32         print(n)
33         [return countdown(n-1)
34         countdown(5)'''
35     #Fixed Code:
36     def countdown(n):
37         if n <= 0:
38             return
39         print(n)
40         countdown(n-1)
41     countdown(5)
42     #Explanation: The original code does not have a base case to stop the recursion, which leads to a maximum recursion depth error when n becomes negative. In the fixed code, we add a base case where n is less than or equal to 0 to prevent this error.
```

The terminal tab at the bottom shows the output of running the code in Python 3:

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7-1.py
[1]
[2]
[1]
[2]
True
[1]
[2]
True
5
4
3
2
1
0
```

## 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.

### EXPLANATION:

The original code tries to access a key "c" that does not exist in the dictionary, which raises a KeyError. In the fixed code, we use the get method of the dictionary, which allows us to specify a default value ("Key not found") to return if the key is not found. This prevents the error and provides a more user-friendly message.

## SCREENSHOT OF GENERATED CODE:

The screenshot shows a VS Code interface with a dark theme. The Explorer sidebar on the left lists files under the 'AIAC' folder, including 'example.py', 'Lab-01.pages', 'Lab-01.pdf', 'Lab-02.docx', 'Lab-02.pages', 'Lab-02.pdf', 'Lab-3-1.pages', 'Lab-3-1.pdf', 'Lab-3-1.py', 'Lab-4-1.pages', 'Lab-4-1.pdf', 'Lab-4-1.py', 'Lab-4-3.pages', 'Lab-4-3.pdf', 'Lab-4-3.py', 'Lab-5-1-4053.pages', 'Lab-5-1-4053.pdf', 'Lab-5-1.py', 'Lab-6-3-4053.pdf', 'Lab-6-3-5053.pages', 'Lab-6-3.py', 'Lab-7-1.py', 'users\_secure.txt', and 'users.txt'. The 'lab-7-1.py' file is open in the editor tab, containing the following code:

```
44     #Task 4
45     #Buggy Code:
46     '''def get_value():
47         [data = {"a": 1, "b": 2}
48         [return data["c"]
49         print(get_value())'''
50     #Fixed Code:
51     def get_value():
52         data = {"a": 1, "b": 2}
53         return data.get("c", "Key not found")
54     print(get_value())
55     #Explanation: The original code tries to access a key "c" that does not exist in the dictionary, which raises a KeyError. In the fixed code, we use the get method
```

The terminal tab at the bottom shows the output of running the code in Python 3:

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7-1.py
2
4
3
2
1
0
[1]
[2]
True
5
4
3
2
1
0
Key not found
<(base) navedahmedshaik@Naveds-MacBook-Air AIAC % []>
```

## 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.

### EXPLANATION:

The original code has an infinite loop because the variable i is never incremented, which means it will always be less than 5. In the fixed code, we add `i += 1` inside the loop to ensure that i is incremented on each iteration, allowing the loop to eventually terminate when i reaches 5.

## SCREENSHOT OF GENERATED CODE:

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows a folder named "AIAC" containing various files like "example.py", "Lab-01.pages", "Lab-02.docx", etc., and several "lab-\*.py" files.
- Code Editor:** Displays the file "lab-7.1.py" with the following content:

```
57 #task 5
58 #buggy Code:
59 '''def loop_example():
60     i = 0
61     while i < 5:
62         print(i)
63
64 #Fixed Code:
65 def loop_example():
66     i = 0
67     while i < 5:
68         print(i)
69         i += 1
70
71 #Explanation: The original code has an infinite loop because the variable i is never incremented, which means it will always be less than 5. In the fixed code, we add i += 1 inside the loop to ensure that i is incremented on each iteration, allowing the loop to eventually terminate when i reaches 5.
```
- Terminal:** Shows the command line output of running the script:

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py
[1] 1234
[2]
True
5
4
3
2
1
0
Key not found
[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.

### EXPLANATION:

The original code tries to unpack a tuple with three values into two variables, which raises a `ValueError`. In the fixed code, we use the `*rest` syntax to capture any extra values in a list called `rest`. This allows us to unpack the first two values into `a` and `b` while still keeping the remaining value in `rest` without causing an error.

## SCREENSHOT OF GENERATED CODE:

```
72 #Task 7
73 #Buggy Code:
74 '''a, b = (1, 2, 3)'''
75 #Fixed Code:
76 # Bug: Wrong unpacking
77 a, b, *rest = (1, 2, 3)
78 print(a) # Output: 1
79 print(b) # Output: 2
80 print(rest) # Output: [3]
81 # Explanation: The original code tries to unpack a tuple with three values into two variables, which raises a ValueError. In the fixed code, we use the *rest syntax.
82
83 |
```

(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py

```
True
5
4
3
2
1
0
Key not found
0
1
2
3
4
1
2
2
[3]
```

(base) navedahmedshaik@Naveds-MacBook-Air AIAC %

## 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.**

### EXPLANATION:

The original code has inconsistent indentation, which leads to an IndentationError. In the fixed code, we ensure that all lines of code within the function have the same level of indentation. This allows the function to execute properly and return the correct result when called.

## SCREENSHOT OF GENERATED CODE:

```
83 #Task 7
84 #Buggy Code:
85 '''def func():
86     [x = 5
87      y = 10
88      ]return x+y'''
89 #Fixed Code:
90 def func():
91     x = 5
92     y = 10
93     return x + y
94 print(func())
95 #Explanation: The original code has inconsistent indentation, which leads to an IndentationError. In the fixed code, we ensure that all lines of code within the function have the same level of indentation.
```

(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py

```
5
4
3
2
1
0
Key not found
0
1
2
3
4
1
2
2
[3]
15
```

(base) navedahmedshaik@Naveds-MacBook-Air AIAC %

## 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

### EXPLANATION:

The original code tries to import a module named "maths", which does not exist in the Python standard library. The correct module name is "math". In the fixed code, we change the import statement to import math, which allows us to use the sqrt function correctly and get the expected output of 4.0 when calculating the square root of 16.

### SCREENSHOT OF GENERATED CODE:

```
97 #Task 8
98 #Buggy Code:
99 """import maths
100 print(maths.sqrt(16))"""
101 #Fixed Code:
102 # Bug: Wrong import
103 import math
104 print(math.sqrt(16))
105 # Explanation: The original code tries to import a module named "maths", which does not exist in the Python standard library. The correct module name is "math". Importing "math" instead of "maths" fixes the bug.
106
107
```

(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py

4  
3  
2  
1  
0  
Key not found  
6  
1  
2  
3  
4  
1  
2  
[3]  
15  
1.8

## 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.

### EXPLANATION:

The original code returns the first number in the list because the return statement is inside the loop, which causes the function to exit after the first iteration. In the fixed code, we initialize a variable `total_sum` to 0 and add each number in the list to `total_sum` within the loop, ensuring that all numbers are summed correctly before returning the total.

## SCREENSHOT OF GENERATED CODE:

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a folder named "AIAC" containing various files like "example.py", "Lab-01.pdf", "Lab-02.docx", etc., and several "lab-\*.py" files.
- Code Editor:** Displays the content of "lab-7.1.py".

```
107 #Task 9
108 #Buggy Code:
109 """def total(numbers):
110     for n in numbers:
111         return n
112     print(total([1,2,3]))"""
113 #Fixed Code:
114 def total(numbers):
115     total_sum = 0
116     for n in numbers:
117         total_sum += n
118     return total_sum
119 print(total([1, 2, 3]))
120 #Explanation: The original code returns the first number in the list because the return statement is inside the loop, which causes the function to exit after the
121 |
```
- Terminal:** Shows the command run in the terminal: `(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py`. The output is:

```
3
2
1
0
Key not found
6
1
2
3
4
1
2
[3]
15
4.0
6
```
- Output Panel:** Shows "Python" as the active interpreter.

## 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.**

## EXPLANATION:

The original code tries to calculate the area using variables length and width that are not defined within the function, which raises a `NameError`. In the fixed code, we modify the function to accept length and width as parameters, allowing us to pass the necessary values when calling the function. This way, we can calculate the area correctly and get the expected output of 15 when calculating the area of a rectangle with length 5 and width 3.

## **SCREENSHOT OF GENERATED CODE:**

The screenshot shows a dark-themed instance of Visual Studio Code (VS Code) with the following details:

- File Explorer (Left):** Shows a tree view of files and folders. The 'AIAC' folder is expanded, containing 'example.py', 'Lab-01.pdf', 'Lab-02.docx', 'Lab-02.pages', 'Lab-02.pdf', 'Lab-3.pdf', 'Lab-3.pages', 'Lab-3.pdf', 'Lab-3.py', 'Lab-4.pdf', 'Lab-4.pages', 'Lab-4.pdf', 'Lab-4.py', 'Lab-4-3.pdf', 'Lab-4-3.py', 'Lab-5-1-4053.pages', 'Lab-5-1-4053.pdf', 'Lab-5-1.py', 'Lab-6-3-4053.pdf', 'Lab-6-3-4053.pages', 'Lab-6-3.py', and 'Lab-7.1.py'. Other files like 'users.secure.txt' and 'users.txt' are also listed.
- Code Editor (Top Center):** Displays the content of 'lab-7.1.py'. The code includes a task 10 section with buggy and fixed versions of a calculate\_area function, and a note about variable scope.
- Terminal (Bottom):** Shows the command `python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py` being run, followed by the output:

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py
2
1
0
Key not found
0
1
2
3
4
5
6
7
[3]
15
4x0
6
15
```
- Status Bar (Bottom Right):** Shows the line number (Ln 132, Col 1), spaces used (Spaces: 4), file type (UTF-8 LF), Python, and the status bar itself.

## 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"
```

- 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.

- Verify with 3 assets

- Corrected code with type handling.

- AI explanation of the fix.

#### **EXPLANATION**

**EXPLANATION:** The original code tries to add an integer (5) and a string ("10"), which raises a TypeError. In the fixed code, we convert the string "10" to an integer using the int() function before performing the addition. This allows us to successfully add the two values and get the expected output of 15.

## SCREENSHOT OF GENERATED CODE:

The screenshot shows a dark-themed version of VS Code. The Explorer sidebar on the left lists files and folders related to AIAC tasks. The main editor area displays a Python file named lab-7.1.py with the following content:

```
133 #Task 1
134 #Buggy Code:
135 '''def add_values():
136     [return 5 + "10"
137     print(add_values())'''
138 #Fixed Code:
139 def add_values():
140     return 5 + int("10")
141 print(add_values())
142 #Explanation: The original code tries to add an integer (5) and a string ("10"), which raises a TypeError. In the fixed code, we convert the string "10" to an integer.
```

The terminal at the bottom shows the output of running the code in Python 3, which results in a `TypeError` because it tries to add a string and an integer. The output is:

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py
1
Key not found
0
1
2
3
4
1
2
[3]
15
4.0
6
15
15
```

## 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.

## SCREENSHOT OF GENERATED CODE:

The screenshot shows a dark-themed version of VS Code. The Explorer sidebar on the left lists files and folders related to AIAC tasks. The main editor area displays a Python file named lab-7.1.py with the following content:

```
144 #Task 12
145 #Buggy Code:
146 '''def combine():
147     [return "Numbers: " + [1, 2, 3]
148     print(combine())'''
149 #Fixed Code:
150 def combine():
151     return "Numbers: " + str([1, 2, 3])
152 print(combine())
153 #Explanation: The original code tries to concatenate a string with a list, which raises a TypeError. In the fixed code, we convert the list [1, 2, 3] to a string.
```

The terminal at the bottom shows the output of running the code in Python 3, which prints the string "Numbers: [1, 2, 3]" to the console.

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py
Key not found
0
1
2
3
4
1
2
[3]
15
4.0
6
15
15
Numbers: [1, 2, 3]
```

## **EXPLANATION:**

The original code tries to concatenate a string with a list, which raises a `TypeError`. In the fixed code, we convert the list `[1, 2, 3]` to a string using the `str()` function before concatenating it with the "Numbers: " string. This allows us to successfully combine the string and the list representation, resulting in the 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())
```

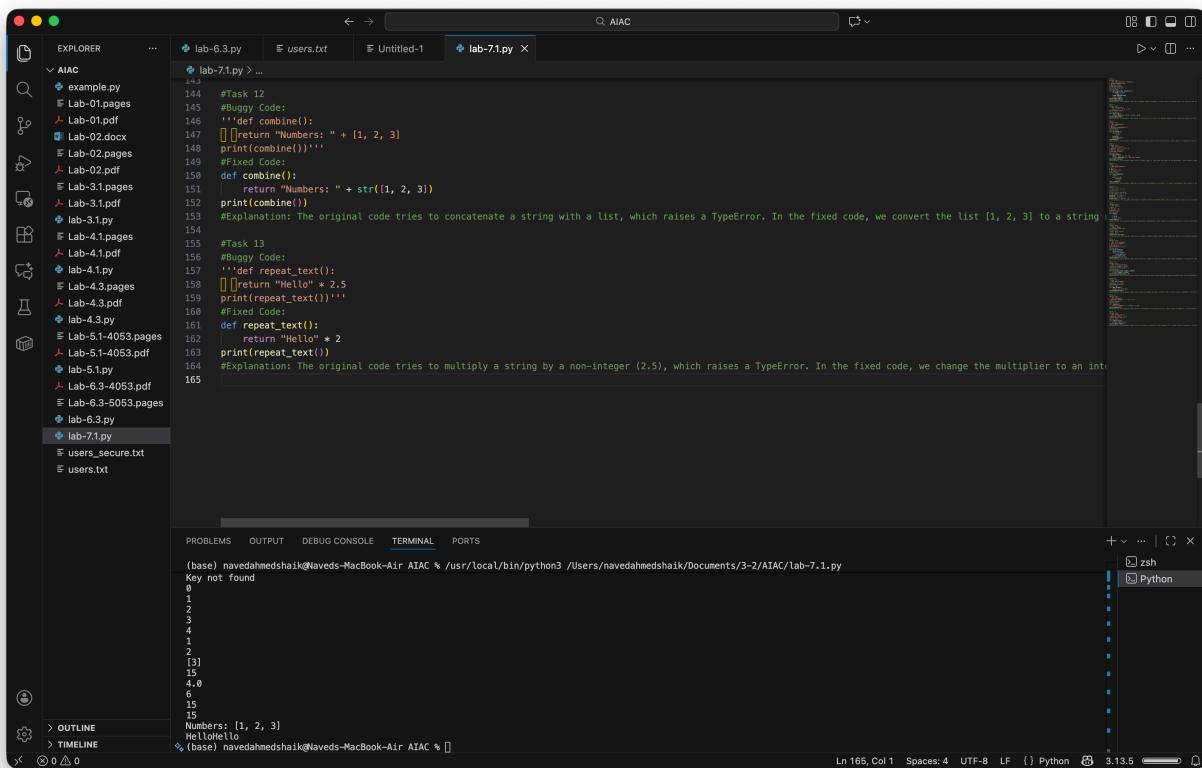
- Requirements:

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

## **EXPLANATION:**

The original code tries to multiply a string by a non-integer (2.5), which raises a `TypeError`. In the fixed code, we change the multiplier to an integer (2), which allows us to repeat the string "Hello" twice successfully, resulting in the output "HelloHello".

## **SCREENSHOT OF GENERATED CODE:**



## 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.

### EXPLANATION:

The original code initializes the variable `value` to `None`, which cannot be added to an integer, resulting in a `TypeError`. In the fixed code, we initialize `value` to `0`, which allows us to perform the addition without any errors. The function now returns `10`, which is the expected output when adding `10` to the initial value of `0`.

### SCREENSHOT OF GENERATED CODE:

```

166 #Task 14
167 #Original Code:
168 '''def compute():
169     [value = None
170      [return value + 10
171      print(compute())'''
172 #Fixed Code:
173 def compute():
174     value = 0
175     return value + 10
176 print(compute())
177 #Explanation: The original code initializes the variable value to None, which cannot be added to an integer, resulting in a TypeError. In the fixed code, we initialize value to 0, allowing the addition to work correctly.
178

```

(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py

```

0
1
2
3
4
1
2
15
Numbers: [1, 2, 3]
HelloHello
15
%
```

Ln 178, Col 1 Spaces: 4 UTF-8 LF () Python 3.13.5

### 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.

### EXPLANATION:

The original code takes user input as strings, and when it tries to add them together, it concatenates the strings instead of performing numerical addition, which is not the intended behavior. In the fixed code, we convert the input strings to floats using the `float()` function before performing the addition. This allows us to correctly sum the two numbers and return the expected numerical result.

## SCREENSHOT OF GENERATED CODE:

The screenshot shows a code editor interface with the following details:

- File Explorer:** On the left, under the "AIAC" folder, there are several files and folders: example.py, Lab-01.pages, Lab-01.pdf, Lab-02.docx, Lab-02.pages, Lab-02.pdf, Lab-3.1.pages, Lab-3.1.pdf, Lab-4.1.pages, Lab-4.1.pdf, Lab-4.1.py, Lab-4.3.pages, Lab-4.3.pdf, Lab-4.3.py, Lab-5.1.py, Lab-6.3-4053.pdf, Lab-6.3-5053.pages, Lab-6.3.py, lab-7.1.py, users\_secure.txt, and users.txt.
- Code Editor:** The main area displays the content of `lab-7.1.py`. The code is as follows:

```
179 #task 15
180 #buggy Code:
181 """def sum_two_numbers():
182     a = input("Enter first number: ")
183     b = input("Enter second number: ")
184     return a + b
185 print(sum_two_numbers())
186 #Fixed Code:
187 def sum_two_numbers():
188     a = float(input("Enter first number: "))
189     b = float(input("Enter second number: "))
190     return a + b
191 print(sum_two_numbers())
192 #Explanation: The original code takes user input as strings, and when it tries to add them together, it concatenates the strings instead of performing numerical addition.
```

- Terminal:** At the bottom, the terminal window shows the execution of the script:

```
(base) navedahmedshaik@Naveds-MacBook-Air AIAC % /usr/local/bin/python3 /Users/navedahmedshaik/Documents/3-2/AIAC/lab-7.1.py
3
4
1
2
[3]
15
4.0
6
15
15
Numbers: [1, 2, 3]
HelloHello
20
Enter first number: 2
Enter second number: 3
5.0
(base) navedahmedshaik@Naveds-MacBook-Air AIAC %
```

- Status Bar:** The bottom right corner shows the status bar with the following information: Ln 193, Col 1 | Spaces: 4 | UTF-8 | LF | {} Python | 3.13.5.