

Lab Assignment-7.5

Name:D.BHARATH KUMAR

Hallticket:2303A51097

Batch:02

Task 1 (Mutable Default Argument – Function Bug)

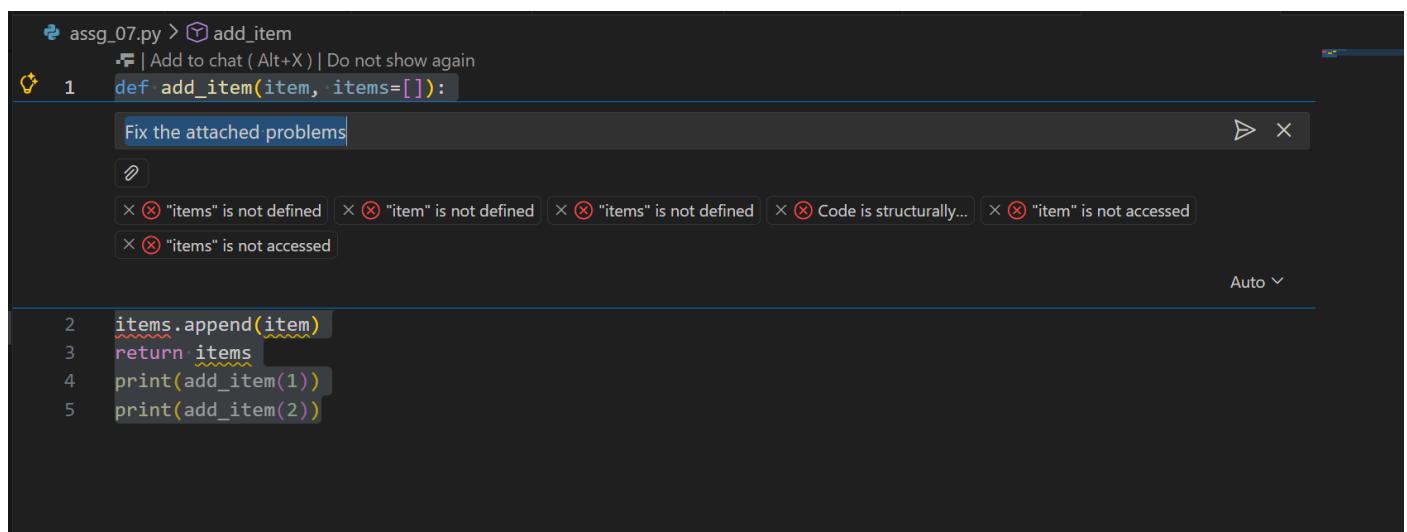
Task: Analyze given code where a mutable default argument causes unexpected behavior. Use AI to fix it.

```
# 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 in VS Code with the file 'assg_07.py' open. The code defines a function 'add_item' with a mutable default argument 'items'. The code is as follows:

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

A tooltip from an AI tool suggests fixing the attached problems, which are:

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

The code editor interface includes a status bar at the bottom right showing "Auto".

```
🐍 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

✓ X

∅ 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))
```

Code:

```
def add_item(item, items=None):
```

```
    if items is None:
```

```
        items = []
```

```
    items.append(item)
```

```
return items

print(add_item(1))

print(add_item(2))
```

output:

```
C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
[1]
[2]
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\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

Screenshots:

The screenshot shows an AI-powered code editor interface. At the top, there is a search bar with the placeholder text "Fix the attached problems floating point prcision issue". Below the search bar are three buttons: a trash icon, an "Expected indented block" button with a red X, and a "Code is structurally..." button with a red X. To the right of these buttons is a "Auto" dropdown menu. The main area displays the following Python code:

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

The screenshot shows the same AI-powered code editor interface after the code has been corrected. The status bar at the bottom now includes "Keep" and "Undo" buttons. The main code area now uses the `math.isclose` function to handle floating-point comparisons:

```
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())
```

Code:

```
import math
```

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

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
True
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\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.

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)
```

```
14
15     def countdown(n):
16         if n <= 0:
17             return
18         print(n)
19         return countdown(n-1)
20     countdown(5)
```

Code:

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

countdown(5)
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
5
4
3
2
1
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> []
```

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:

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

Code:

```
def get_value():

    data = {"a": 1, "b": 2}

    return data.get("c", "Key not found")

print(get_value())
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
Key not found
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\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.

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()
```

Code:

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

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
0
1
2
3
4
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> 
```

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)
```

Code:

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

```
print(a, b)
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
1 2
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\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.

Screenshots:

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

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

Code:

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

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
15
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> []
Ln 45, Col 14  Spaces: 4  UTF-8  CRLF  {} Python  3.13.5
```

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:

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

- Input field: "correct the code"
- Status bar: "Auto" dropdown
- Message bar: "Import "maths" could not be..." (with a warning icon)
- Code area:

```
46
47 import maths
48 print(maths.sqrt(16))
49
50
```
- Toolbars: "Add to chat (Alt+X)" and "Do not show again"

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

Code:

```
import math
```

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

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
4.0
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> []
```

Task 9 (Unreachable Code – Return Inside Loop)

Task: Analyze given code where a return inside a loop prevents full

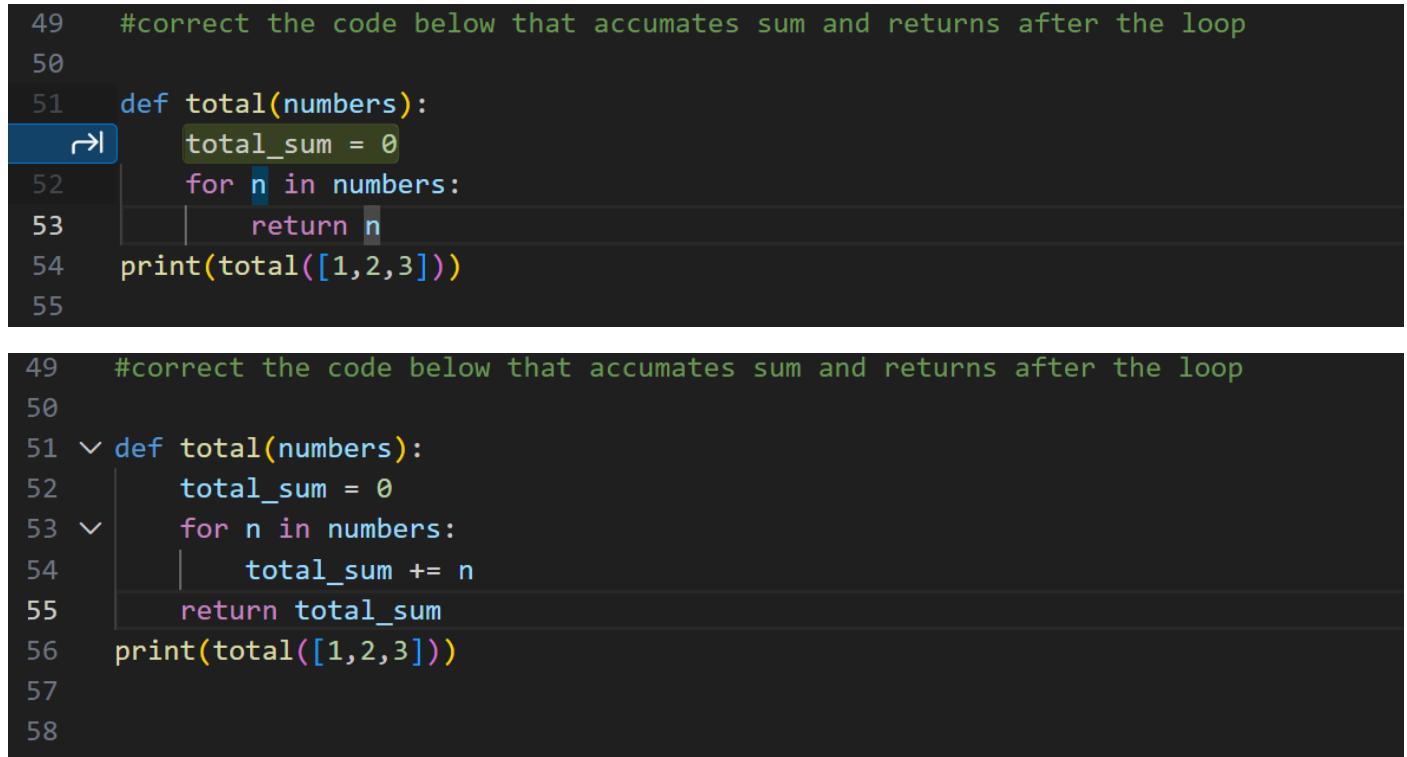
iteration. Use AI to fix it.

```
# 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):  
    ↳     total_sum = 0  
52     for n in numbers:  
53         return n  
54 print(total([1,2,3]))  
55  
  
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         total_sum += n  
55     return total_sum  
56 print(total([1,2,3]))  
57  
58
```

Code:

```
def total(numbers):  
    total_sum = 0  
    for n in numbers:  
        total_sum += n  
    return total_sum  
  
print(total([1,2,3]))
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Des  
ktop/AIAC/assg_07 (1).py"  
6  
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> [ ]  
In 51, Col 1 Spaces: 4 UTF-8 CR/LF { } Python 3.13.5
```

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:

The screenshot shows a code editor interface with a dark theme. At the top left, the number '58' is displayed. Below it, a blue bar contains a right-pointing arrow icon followed by the text 'from turtle import width'. The main code area starts at line 59, which defines a function 'calculate_area' that returns the product of 'length' and 'width'. Both 'length' and 'width' are underlined in yellow, indicating they are undefined variables. Line 61 prints the result of the function call. Line 62 is a blank line.

```
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
```

Code:

```
from turtle import width

def calculate_area(length, width):

    return length * width

print(calculate_area(5, 10)) # Example values for length and width
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
50
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> []
Ln 60, Col 1  Spaces: 4  UTF-8  CRLF  {} Python  3.13.5
```

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.

Screenshots:

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

```
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())
```

Code:

```
def add_values():

    return 5 + int("10") # The error occurs because we are trying to add an integer (5) and a string ("10"). To
fix this, we need to convert the string "10" to an integer using the int() function before performing the
addition.

print(add_values())
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & c:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
15
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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:

```

Modify selected code
∅ Add Context...
✓ ×
Keep (Ctrl+Enter)

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

```



```

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

```

Output:

```

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
Numbers: 1, 2, 3
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

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 a "Keep Undo" button. Below the toolbar, the code is displayed in a dark-themed editor. A tooltip is overlaid on the line "def repeat_text(text=None):". The tooltip contains the following text:
- 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

```
76
    | Modify selected code
    | ⚙ Add Context...
    | Auto
    | Keep Undo
76
def repeat_text():
    return "Hello" * 2
    | Add to chat (Alt+X) | Do not show again
77
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
```

A screenshot of a code editor interface showing the final version of the code after fixing the NoneType error. The code is identical to the one shown in the tooltip above.

```
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
```

Code:

```
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

print(repeat_text())
```

output:

```
Number 3. 1, 2, 3
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
HelloHello
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> []
```

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.

Screenshots:

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

```
87
88     def sum_two_numbers():
89         a = input("Enter first number: ")
90         b = input("Enter second number: ")
91
92         # input() always returns a string, even if the user enters numbers
93         # Attempting to add two strings will concatenate them instead of summing
94         # Example: "5" + "3" = "53" (string concatenation, not addition)
95         a = int(input("Enter first number: "))
96         b = int(input("Enter second number: "))
97
98         # Convert strings to integers using int() before performing arithmetic
99         return a + b
100
101 print(sum_two_numbers())
```

Annotations explain the behavior of the code:

- Line 88: A note above the code says: "input() always returns a string, even if the user enters numbers".
- Line 90: A note says: "Attempting to add two strings will concatenate them instead of summing".
- Line 94: A note says: "Example: "5" + "3" = "53" (string concatenation, not addition)".
- Line 95: A note says: "# Convert strings to integers using int() before performing arithmetic".

```
87
88 def sum_two_numbers():
89     # input() always returns a string, even if the user enters numbers
90     # Attempting to add two strings will concatenate them instead of summing
91     # Example: "5" + "3" = "53" (string concatenation, not addition)
92     a = int(input("Enter first number: "))
93     b = int(input("Enter second number: "))
94     # Convert strings to integers using int() before performing arithmetic
95     return a + b
96
97 print(sum_two_numbers())
98
```

Code:

```
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

print(sum_two_numbers())
```

output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:/Users/BHARATH/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_07 (1).py"
Enter first number: 10
Enter second number: 20
30
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

In 88 Col 1 Spaces: 4 UTE-8 CRLE {} Python 3.13.5