

AI Assisted Coding Assignment- 7.5

| 2303A51543 BT: 29 VELDI.HRUTHIKA 10.02.2026



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.

```
#2303a51543  
#Task 1  
# Bug: Mutable default argument  
def add_item(item, items=[]):  
    items.append(item)  
    return items  
print(add_item(1))  
print(add_item(2))  
# Fix: Use None as default argument  
def add_item(item, items=None):  
    if items is None:  
        items = []  
    items.append(item)  
    return items  
print(add_item(1))  
print(add_item(2))
```

```
(2).py | Ass_4.5(3).py | Ass_5.5.py | Ass_6.5.py | Ass_7.5.py  
Ass_7.5.py > ...  
1 #Task 1  
2 # Bug: Mutable default argument  
3 def add_item(item, items=[]):  
4     items.append(item)  
5     return items  
6 print(add_item(1))  
7 print(add_item(2))  
8 # Fix: Use None as default argument  
9 def add_item(item, items=None):  
10    if items is None:  
11        items = []  
12        items.append(item)  
13        return items  
14  
15  
16
```



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

```
#Task -2  
# Bug: Floating point precision issue  
def check_sum():  
    return (0.1 + 0.2) == 0.3  
print(check_sum())  
# Fix: Corrected function  
def check_sum():  
    return abs((0.1 + 0.2) - 0.3) < 1e-9  
print(check_sum())
```

```
● /Ass_7.5.py > check_sum
18     # Bug: Floating point precision issue
19     def check_sum():
20         return (0.1 + 0.2) == 0.3
21     print(check_sum())
22     # Fix: Corrected function
23     def check_sum():
24         return abs((0.1 + 0.2) - 0.3) < 1e-9
25 →|  print(check_sum())
26
27
```



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.

```
#Task -3
# Bug: No base case
#def countdown(n):
#    print(n)
#    return countdown(n-1)
#countdown(5)
# Fix: Correct recursion with stopping condition.
def countdown(n):
    if n <= 0:
        print("Blast off!")
    else:
        print(n)
        countdown(n-1)
countdown(5)
```

```

26
27 #Task -3
28 # Bug: No base case
29 def countdown(n):
30     print(n)
31     return countdown(n-1)
32 countdown(5)
33 # Fix: Correct recursion with stopping condition.
34 def countdown(n):
35     if n <= 0:
36         print("Blast off!")
37     else:
38         print(n)
39         countdown(n-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.

```
#Task -4
# Bug: Accessing non-existing key
# Fix: Corrected with .get() or error handling.
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")
print(get_value())
```

A screenshot of the Visual Studio Code interface. The top part shows a code editor with a dark theme. A tooltip is displayed over line 49, which contains the code 'def get_value_safe():'. The tooltip shows the completed function definition: 'def get_value_safe(): data = {"a": 1, "b": 2} return data.get("c", "Key not found")'. The line number 49 has a blue background and a checkmark icon. Below the code editor is a navigation bar with tabs: PROBLEMS, OUTPUT, TERMINAL (underlined), PORTS, and DEBUG CONSOLE.

```
41
42     #Task -4
43     # Bug: Accessing non-existing key
44     def get_value():
45         data = {"a": 1, "b": 2}
46         return data["c"]
47     print(get_value())
48     # Fix: Corrected with .get() or error handling.
49 ✓     def get_value_safe():
50         data = {"a": 1, "b": 2}
51         return data.get("c", "Key not found")
    print(get_value_safe())
```

A screenshot of the Visual Studio Code interface, similar to the one above but with a different view. It shows the same code editor with the corrected function. Below the code editor is a terminal window. The terminal output shows the result of running the code: 'Key not found'. The terminal tab is underlined, and the status bar at the bottom right shows 'Python'.

```
42     #Task -4
43     # Bug: Accessing non-existing key
44     # Fix: Corrected with .get() or error handling.
45     def get_value():
46         data = {"a": 1, "b": 2}
47         return data.get("c", "Key not found")
48     print(get_value())
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

Key not found
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>



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.
```

```
#Task -5
# Bug: Infinite loop
#def loop_example():
#    i = 0
#    while i < 5:
#        print(i)
#Fix: Corrected loop increments i.
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1
loop_example()
```

```
.9
.0 #Task -5
.1 # Bug: Infinite loop
.2 def loop_example():
.3     i = 0
.4     while i < 5:
.5         |     print(i)
.6 #Fix: Corrected loop increments i.
.7 def Loop_example():
.8     i = 0
.9     while i < 5:
.0         print(i)
.1         i += 1
```

The screenshot shows a code editor window with a dark theme. At the top, there's a tab labeled 'Ass_7.5.py > ...'. Below it is the Python code:

```
51 # Bug: Infinite loop
52 #def loop_example():
53 #    i = 0
54 #    while i < 5:
55 #        print(i)
56 #Fix: Corrected loop increments i.
57 def loop_example():
58     i = 0
59     while i < 5:
60         print(i)
61         i += 1
62 loop_example()
```

Below the code are tabs for 'PROBLEMS', 'OUTPUT', 'TERMINAL', 'PORTS', and 'DEBUG CONSOLE'. The 'TERMINAL' tab is selected, showing the command and its output:

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppD
on.exe c:/Users/hruth/OneDrive/Desktop/A.I.AC/Ass_7.5.py
0
1
2
3
4
```

The terminal prompt then changes to:

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
```



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.

```
#Task -6
# Bug: Wrong unpacking
#a, b = (1, 2, 3)
#Fix: Correct unpacking or using _ for extra values.
a, b, _ = (1, 2, 3)
```

```
#Task -6
# Bug: Wrong unpacking
a, b = (1, 2, 3)
#Fix: Correct unpacking or using _ for extra values.
a, b, _ = (1, 2, 3)
```



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.

```
#Task -7
# Bug: Mixed indentation
#def func():
#    x = 5
#    y = 10
#    return x+y
#Fix: Consistent indentation
def func():
    x = 5
    y = 10
    return x + y
print(func())
```

```
Ass_7.5.py > ...
71 # Bug: Mixed indentation
72 def func():
73     x = 5
74     y = 10
75     return x+y
76 #Fix: Consistent indentation
77 def func():
78     x = 5
79     y = 10
80     return x + y
```

```
69
70 #Task -7
71 # Bug: Mixed indentation
72 #def func():
73 #     x = 5
74 #         y = 10
75 #     return x+y
76 #Fix: Consistent indentation
77 def func():
78     x = 5
79     y = 10
80     return x + y
81 print(func())
82
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

...

○ 4
15

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>



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

```
#Task-8
# Bug: Wrong import
#import maths
#print(maths.sqrt(16))
#Fix:Corrected to import math
import math
print(math.sqrt(16))
```

```
#Task-8
# Bug: Wrong import
import maths
print(maths.sqrt(16))
#Fix:Corrected to import math
import math
print(math.sqrt(16))
```



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.

```
#Task -9
# Bug: Early return inside loop
#def total(numbers):
#    for n in numbers:
#        return n
#print(total([1,2,3]))
```

```
#Fix: Corrected code accumulates sum and returns after loop.
def total(numbers):
    sum = 0
    for n in numbers:
        sum += n
    return sum
print(total([1,2,3]))
```

```
#Task -8
# Bug: Early return inside loop
def total(numbers):
    for n in numbers:
        return n
print(total([1,2,3]))
#Fix: Corrected code accumulates sum and returns after loop.
def total(numbers):
    sum = 0
    for n in numbers:
        sum += n
    return sum
```



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.

```
#Task -10
# Bug: Using undefined variable
#def calculate_area():
#    return length * width
```

```

#print(calculate_area())

#Fix: • Corrected code with parameters.AI explanation of the bug..• Ask AI to identify th
e missing variable definition..• Fix the bug by defining length and width as parameters..•
Add 3 assert test cases for correctness.

def calculate_area(length, width):
    return length * width
print(calculate_area(5, 10))
# Test cases
assert calculate_area(5, 10) == 50
assert calculate_area(3, 4) == 12
assert calculate_area(7, 2) == 14

```

```

ass_7.5.py > ...
#Task -10
# Bug: Using undefined variable
def calculate_area():
    return length * width
print(calculate_area())

#Fix: • Corrected code with parameters.AI explanation of the bug..• Ask AI to i
def calculate_area(length, width):
    return length * width
# Test cases
assert calculate_area(5, 10) == 50
assert calculate_area(3, 4) == 12
assert calculate_area(7, 2) == 14

```

```

109  #print(calculate_area())
110
111 #Fix: • Corrected code with parameters.AI explanation of the bug..• Ask AI to identify the missing variab
112 def calculate_area(length, width):
113     return length * width
114 print(calculate_area(5, 10))
115 # Test cases
116 assert calculate_area(5, 10) == 50
117 assert calculate_area(3, 4) == 12
118 assert calculate_area(7, 2) == 14
119
120
121
122
123

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
Python + × ⌂ ⌂ ... | ⌂ ×
50
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
Ln 114, Col 29 Spaces: 4 UTF-8 CRLF {} Python 3.12.3 ⌂ Go Live ⌂ Prettier ⌂

```

AI Explanation of the Bug:

The bug occurs because the function `calculate_area()` is using two variables (`length` and `width`) that are not defined within the function. Python needs these variables to be passed to the function as arguments, but they are missing.

How the Fix Works:

To fix this:

1. We define `length` and `width` as parameters in the function.
2. Then, we pass values for these parameters when calling the function.
3. Finally, we added 3 `assert` test cases to make sure the function works correctly for different input values.



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.

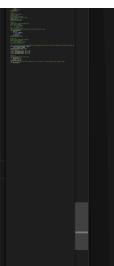
Expected Output #6:

- Corrected code with type handling.
- AI explanation of the fix.

Successful test validation.

```
#Task-11
# Bug: Adding integer and string
def add_values():
    #     return 5 + "10"
#print(add_values())
#Fix::• Fix the code by type conversion (e.g., int("10") or str(5)).Verify with 3 assert
cases.
def add_values():
    return 5 + int("10")
print(add_values())
```

```
18     assert calculate_area(2) == 14
19
20 #Task-11
21 # Bug: Adding integer and string
22 def add_values():
23     #     return 5 + "10"
24     print(add_values())
25 #Fix::• Fix the code by type conversion (e.g., int("10") or str(5)).Verify with 3 assert
cases.
26 def add_values():
27     return 5 + int("10")
28
29
```



AI Explanation of the Bug:

The bug happens because Python doesn't allow adding an integer (5) and a string ("10") directly. These two types are incompatible for addition.

In Python, an integer is a number, and a string is a sequence of characters. When you try to add them together, Python raises a `TypeError` because it doesn't know how to combine a number and a sequence of characters.



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

```
#Task- 12
# Bug: Adding string and list
#def combine():
#    return "Numbers: " + [1, 2, 3]
#print(combine())

#Fix::• Fix using conversion (str([1,2,3]) or ".join()).Verify with 3 assert cases.
def combine():
    return "Numbers: " + str([1, 2, 3])
print(combine())
```

```
# Bug: Adding string and list
def combine():
    return "Numbers: " + [1, 2, 3]
print(combine())

#Fix::• Fix using conversion (str([1,2,3]) or ".join().)
def combine():
    return "Numbers: " + str([1, 2, 3])
```

AI Explanation of the Bug:

The bug occurs because Python doesn't allow adding a string ("Numbers: ") and a list ([1, 2, 3]) directly. In Python, a string is a sequence of characters, and a list is a collection of items. Since these are two different data types, trying to

add them together directly will raise a `TypeError`.



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.

```
#Task -13
# Bug: Multiplying string by float
#def repeat_text():
#    return "Hello" * 2.5
#print(repeat_text())

#Fix by converting float to int.
def repeat_text():
    return "Hello" * int(2.5)
print(repeat_text())
```

```
0
1 #Task -13
2 # Bug: Multiplying string by float
3 def repeat_text():
4     return "Hello" * 2.5
5 print(repeat_text())
6
7 #Fix by converting float to int.
8 →  def repeat_text():
9     return "Hello" * int(2.5)
0
1
2
3
```

AI Explanation of the Bug:

The bug occurs because Python doesn't allow multiplying a string ("Hello") by a float (2.5). String multiplication only works with an integer value, which determines how many times the string should be repeated. When you try to multiply by a float, Python raises a `TypeError`, as it doesn't know how to handle fractional repetitions of a string.

How the Fix Works:

To fix this:

1. We convert the float 2.5 to an integer using `int(2.5)`.
2. This converts 2.5 into 2, so the string "Hello" will be repeated 2 times.
3. Now, we can multiply the string by an integer without causing any errors.



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.

```
#Task-14
# Bug: Adding None and integer
#def compute():
#    value = None
#    return value + 10

#print(compute())
#• Explain why NoneType cannot be added.
#    Fix by assigning a default value.

def compute():
    value = 0 # Default value instead of None
    return value + 10
print(compute())
```

```
51
52     #Task-14
53     # Bug: Adding None and integer
54     def compute():
55         value = None
56         return value + 10
57
58     print(compute())
59     #• Explain why NoneType cannot be added.
60     # Fix by assigning a default value.
61
62 ↗ def compute():
63     value = 0 # Default value
64     return value + 10
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

AI Explanation of the Bug:

The bug happens because `None` is a special data type in Python, representing the absence of a value. When you try to add `None` to an integer (like `None + 10`), Python raises a `TypeError` because `None` cannot be directly combined with other data types like integers. The operation is undefined, as `None` is not considered a valid operand for arithmetic.

How the Fix Works:

To fix this:

1. We assign a default value, such as `0`, to `value` instead of leaving it as `None`.
2. This ensures that when we perform the addition (`value + 10`), both operands are valid (an integer plus another integer).



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:

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

```
#Task-15
# 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())
#Fix using int() conversion.
def sum_two_numbers():
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    return a + b
print(sum_two_numbers())
```

```
167  #Task-15
168  # Bug: Input remains string
169  def sum_two_numbers():
170  |   a = input("Enter first number: ")
171  |   b = input("Enter second number: ")
172  |   return a + b
173
174  print(sum_two_numbers())
175  #Fix using int() conversion.
176  def sum_two_numbers():
177      a = int(input("Enter first number: "))
178      b = int(input("Enter second number: "))
179      return a + b
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

The screenshot shows a code editor in VS Code with Python code. The code defines a function to sum two numbers, but it uses `input()` which returns strings, causing a bug where the numbers are concatenated instead of added. The terminal shows the execution of the script and the user's input.

```
166
167 #Task-15
168 # Bug: Input remains string
169 #def sum_two_numbers():
170 #    a = input("Enter first number: ")
171 #    b = input("Enter second number: ")
172 #    return a + b
173
174 #print(sum_two_numbers())
175 #Fix using int() conversion.
176 def sum_two_numbers():
177     a = int(input("Enter first number: "))
178     b = int(input("Enter second number: "))
179     return a + b
180 print(sum_two_numbers())
181
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/OneDrive/Desktop/A.I.AC/Ass_7.5.py

Enter first number: 12
Enter second number: 73
85

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>

Ln 175, Col 20 Spaces: 4 UT

AI Explanation of the Bug:

The bug happens because the `input()` function in Python always returns a string, regardless of what the user types. So, when you try to add two inputs (`a + b`), Python is adding two strings together, not numbers. This results in string concatenation instead of numerical addition.

For example, if the user enters `3` and `5`, the code would treat them as strings (`"3"` and `"5"`) and concatenate them into `"35"` rather than adding them as numbers.