

AI ASSISTED -CODING

ASSIGNMENT 7.5

2303A51918

RUDROJU RUPA SRI

BATCH-30

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 screenshot shows the VS Code interface with the following details:

- Editor:** The main editor window displays the code for Task 1. It highlights the mutable default argument `items=[]` and suggests a fix: "Fix: Use None as default argument". The corrected code uses `items=None` and handles the case where `items` is `None` by initializing it to an empty list.
- Terminal:** The terminal shows the output of running the corrected script: `python ASS-7.5.py`. The output is:

```
[1]
[1, 2]
[1]
[2]
```
- CHAT:** A sidebar titled "FIXING INFINITE RECURSION IN COUNT..." shows a snippet of code with a comment "Run the file with:" followed by the command `python ASS-7.5.py`.
- Bottom Status Bar:** Shows the current file is "ASS-7.5.py", encoding is "UTF-8", and Python version is "Python 3.13 (64-bit)".

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

The screenshot shows the VS Code interface with two tabs open: ASS-6.5.py and ASS-7.5.py. The ASS-7.5.py tab contains the following code:

```
16
17  #Task -2
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())
```

The AI Chat sidebar suggests the following fix:

```
← FIXING INFINITE RECURSION IN COUNT...
print(n)
countdown(n-1)

countdown(5)
```

Run the file with:

```
python ASS-7.5.py
```

Expected output:

- 1st task prints lists
- True for task 2
- Then:
5
4
3
2
1
Blast off!

The Terminal pane shows the command "python ASS-7.5.py" being run, and the output is:

```
PS C:\Users\HP\Desktop\AI> & "C:\Program Files\Python313\python.exe" c:/Users/HP/Desktop/AI/ASS-7.5.py
[2]
PS C:\Users\HP\Desktop\AI> & "C:\Program Files\Python313\python.exe" c:/Users/HP/Desktop/AI/ASS-7.5.py
[1]
[1, 2]
[1]
[2]
False
True
○ PS C:\Users\HP\Desktop\AI>
```

Bottom status bar: Ln 25, Col 19 Spaces: 4 UTF-8 CRLF { } Python Python 3.13 (64-bit) (→) Go Live

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.

The screenshot shows the Visual Studio Code interface with the following details:

- Code Editor:** Two tabs are open: "ASS-7.5.py" and "Untitled-1". The "ASS-7.5.py" tab contains Python code for a countdown function. The AI assistant has provided a fix for the base case, resulting in the following code:


```

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)
40 countdown(5)
      
```
- Terminal:** The terminal shows the command "python ASS-7.5.py" being run. The output is:


```

PS C:\Users\HP\Desktop\AI> & "C:\Program Files\Python313\python.exe" c:/Users/HP/Desktop/AI/ASS-7.5.py
[2]
False
True
5
4
3
2
1
Blast off!
      
```
- AI Assistant Panel:** A sidebar titled "FIXING INFINITE RECURSION IN COUNT..." shows the original buggy code and the corrected code. It also includes instructions to "Run the file with: python ASS-7.5.py" and a list of "Expected output" items.
- Bottom Status Bar:** Shows the current file is "ASS-7.5.py", encoding is "UTF-8", and Python version is "Python 3.13 (64-bit)".

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.

ASS-7.5.py

```

41
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 DEBUG CONSOLE TERMINAL PORTS

Key not found
PS C:\Users\HP\Desktop\AI>

CHAT

← FIXING INFINITE RECURSION IN COUNT...

```

print(n)
countdown(n-1)

countdown(5)

```

Run the file with:

`python ASS-7.5.py`

Expected output:

- 1st task prints lists
- `True` for task 2
- Then:


```

5
4
3
2
1
Blast off!

```

Ln 40, Col 13 Spaces: 4 UTF-8 CRLF { } Python 8 Python 3.13 (64-bit) (i) Go Live 🔍

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.

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

- File Explorer:** Shows two files: ASS-6.5.py and ASS-7.5.py.
- Code Editor:** ASS-7.5.py contains the following code:


```

50  #Task -5
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()
      
```
- Terminal:** The terminal shows the output of running `python ASS-7.5.py`. It prints the numbers 5, 4, 3, 2, 1, followed by "Blast off!".
- Chat:** A sidebar titled "CHAT" shows a snippet of code for fixing infinite recursion in a `countdown` function.
- Run Configuration:** A button labeled "Run the file with: python ASS-7.5.py" is present.
- Expected Output:** A list of bullet points:
 - 1st task prints lists
 - True for task 2
 - Then:
 - 5
 - 4
 - 3
 - 2
 - 1
 - Blast off!

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.

The screenshot shows the Visual Studio Code interface. In the top left, there are tabs for 'ASS-6.5.py', 'ASS-7.5.py X', and 'Untitled-1'. The main code editor pane contains Python code with several comments and annotations. The terminal pane at the bottom shows the output of running 'ASS-7.5.py' in a PowerShell window. The output includes numerical values and the text 'Blast off!'. To the right of the terminal is the AI assistant pane, which displays a fix for infinite recursion and provides instructions for running the file.

```

def loop_example():
    print(i)
    i += 1
loop_example()

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + v ... | [] X
PS C:\Users\HP\Desktop\AI> & "C:\Program Files\Python313\python.exe" c:/Users/HP/Desktop/AI/ASS-7.5.py
3
2
1
Blast off!
Key not found
0
1
2
3
4

```

CHAT + v ⚡ ... | [] X
← FIXING INFINITE RECURSION IN COUNT...
print(n)
countdown(n-1)
countdown(5)

Run the file with:
python ASS-7.5.py

Expected output:

- 1st task prints lists
- True for task 2
- Then:


```

5
4
3
2
1
Blast off!
```

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.

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

- Editor:** ASS-7.5.py (active tab), ASS-6.5.py, Untitled-1.
- Code Content:** ASS-7.5.py contains code with mixed indentation bugs. The AI suggestion in the right panel is to fix it with consistent indentation.
- Terminal:** Powershell and PowerShell tabs are visible.
- Output:** Shows the command "python ASS-7.5.py" and its expected output: 5, 4, 3, 2, 1, Blast off!
- Status Bar:** Ln 81, Col 14, Spaces: 4, UTF-8, CRLF, Python 3.13 (64-bit).

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

The screenshot shows the Visual Studio Code interface. The code editor has three tabs: ASS-6.5.py, ASS-7.5.py (active), and Untitled-1. The ASS-7.5.py tab contains Python code with a bug where it imports 'maths' instead of 'math'. The terminal below shows a PowerShell prompt at C:\Users\HP\Desktop\AI>. The Chat panel on the right shows a conversation about fixing infinite recursion in a countdown function. The terminal output shows the expected countdown from 5 to 1.

```
82
83 #Task-8
84 # Bug: Wrong import
85 #import maths
86 #print(maths.sqrt(16))
87 #Fix:Corrected to import math
88 import math
89 print(math.sqrt(16))
```

```
← FIXING INFINITE RECURSION IN COUNT...
print(n)
countdown(n-1)

countdown(5)

Run the file with:
python ASS-7.5.py

Expected output:
• 1st task prints lists
• True for task 2
• Then:
5
4
3
2
1
Blast off!
```

```
4.0
PS C:\Users\HP\Desktop\AI>
```

```
Ln 89, Col 21 Spaces: 4 UTF-8 CRLF { } Python Python 3.13 (64-bit) (Go Live) ⚡
```

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 the Visual Studio Code interface with several windows open:

- Editor:** Shows the file `ASS-7.5.py` with code related to a bug fix for a recursive function.
- Terminal:** Shows the command `PS C:\Users\HP\Desktop\AI>`.
- Output:** Shows the output of the code execution, including the expected output of the countdown function.
- Chat:** Shows a conversation about fixing infinite recursion in a `countdown` function.
- Task List:** Shows a task titled "Task 10 (Name Error – Undefined Variable)" with instructions to analyze given code where a variable is used before being defined.

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.

The screenshot shows a VS Code interface with several tabs open: ASS-6.5.py, ASS-7.5.py (active), and Untitled-1. The ASS-7.5.py tab contains the following code:

```

105 #Task -10
106 # Bug: Using undefined variable
107 # Fix: Corrected code with parameters.
108 def calculate_area(length, width):
109     return length * width
110 print(calculate_area(5, 10))
111 # Test cases
112 assert calculate_area(5, 10) == 50
113 assert calculate_area(3, 4) == 12
114 assert calculate_area(7, 2) == 14
115
116

```

The right pane displays the AI's fix for an infinite recursion bug in COUNTDOWN:

```

print(n)
countdown(n-1)

countdown(5)

```

Run the file with:

```
python ASS-7.5.py
```

Expected output:

- 1st task prints lists
- True for task 2
- Then:

```

5
4
3
2
1
Blast off!

```

Below the terminal, the status bar shows: Ln 115, Col 1 Spaces: 4 UTF-8 { } Python Python 3.13 (64-bit) (r) Go Live

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.

The screenshot shows the Visual Studio Code interface. In the top left, there are three tabs: ASS-6.5.py, ASS-7.5.py X, and Untitled-1. The ASS-7.5.py tab is active, displaying Python code. The code defines a function add_values() that adds an integer 5 and a string "10". It includes test cases using assert statements. The code editor has syntax highlighting for Python. Below the editor is a terminal window showing the command 'python ASS-7.5.py' and its output, which is a list of numbers from 5 down to 1. To the right of the terminal is an AI panel titled 'FIXING INFINITE RECURSION IN COUNT...'. It contains the corrected code where the addition is fixed using int("10"). Below the code is a section titled 'Run the file with:' containing the command 'python ASS-7.5.py'. Another section titled 'Expected output:' lists the expected output: '1st task prints lists', 'True for task 2', and 'Then: 5, 4, 3, 2, 1'. At the bottom of the AI panel, there's a summary: 'Explore and understand your code' with options 'Ask' and 'Auto'.

```

115
116  #Task-11
117  # Bug: Adding integer and string
118  # Fix: Fix the code by type conversion (e.g., int("10") or str(5))
119  def add_values():
120      return 5 + int("10")
121  print(add_values())
122  # Test cases
123  assert add_values() == 15
124  assert 5 + int("10") == 15
125  assert int("10") + 5 == 15
126
127
128

```

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

The screenshot shows a VS Code interface with two tabs open: ASS-6.5.py and ASS-7.5.py. The ASS-7.5.py tab contains Python code for Task-12, which includes a bug fix for concatenating a string and a list. The AI Chat sidebar provides an explanation of the bug and offers a fixed version of the code. The terminal below shows the code being run and the output 'Numbers: [1, 2, 3]'.

```

127 #Task- 12
128 # Bug: Adding string and list
129 def combine():
130     return "Numbers: " + [1, 2, 3]
131     #print(combine())
132 # Fix: Fix using conversion (str([1,2,3]) or ".join()). Verify
133 def combine():
134     return "Numbers: " + str([1, 2, 3])
135     print(combine())
136 # Test cases
137 assert combine() == "Numbers: [1, 2, 3]"
138 assert "Numbers: " + str([1, 2, 3]) == "Numbers: [1, 2, 3]"
139 assert combine() != "Numbers: 1 2 3"
140
141

```

AI Chat Suggestion:

```

#Task-12
# Bug: Adding string and list
def combine():
    # return "Numbers: " + [1, 2,
    #print(combine())
# Fix: Fix using conversion (str([1,2,3]) or ".join())
def combine():
    return "Numbers: " + str([1,
print(combine())
# Test cases
assert combine() == "Numbers: [1,
assert "Numbers: " + str([1, 2,
assert combine() != "Numbers: 1 "

```

Explanation of the fix:

- **Bug:** You cannot concatenate a string with a list directly.

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.

The screenshot shows a VS Code interface with several tabs open: ASS-6.5.py, ASS-7.5.py (active), and Untitled-1. The ASS-7.5.py tab contains the following code:

```
140
141     #Task -13
142     # Bug: Multiplying string by float
143     # Fix: Convert float to int.
144     def repeat_text():
145         return "Hello" * int(2.5)
146     print(repeat_text())
147     # Test cases
148     assert repeat_text() == "HelloHello"
149     assert "Hello" * int(2.5) == "HelloHello"
150     assert repeat_text() != "HelloHelloHello"
151
152
153
154
```

The terminal below shows the output: "HelloHello". The status bar indicates the file is saved and shows Python 3.13 (64-bit).

The right side of the interface has an AI-powered explanation for the bug:

CHAT
← FIXING STRING AND LIST CONCATENATION
assert combine() != "Numbers: 1 :"

Explanation of the fix:

- Bug: You cannot concatenate a string with a list directly using + operator (type mismatch)
- Solution: Convert the list to a string using str([1, 2, 3]), which produces "[1, 2, 3]"

• Test cases verify:

1. The output equals the expected string with list representation
2. Manual concatenation produces the same result
3. The output is NOT the space-separated version (confirming str() conversion, not join())

You can add this code to the end of your ASS-7.5.py file.

Explore and understand your code

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.

The screenshot shows a VS Code interface with two tabs open: `ASS-6.5.py` and `ASS-7.5.py`. The `ASS-7.5.py` tab contains the following code:

```
152
153     #Task-14
154     # Fixed function with default value
155     def compute():
156         value = 0      # Default value instead of None
157         return value + 10
158     # Run the function
159     print(compute())
160     # Validate using asserts
161     assert compute() == 10
162     print("All tests passed ✓")
163
164
165
166
```

The terminal below shows the output of running the script:

```
PS C:\Users\HP\Desktop\AI> & "C:\Program Files\Python313\python.exe" c:/Users/HP/Desktop/AI/ASS-7.5.py
4
15
4.0
6
50
15
Numbers: [1, 2, 3]
HelloHello
10
All tests passed ✓
PS C:\Users\HP\Desktop\AI>
```

A floating AI panel on the right provides an explanation of the bug and its fix:

Explanation of the fix:

- **Bug:** You cannot concatenate a string with a list directly using `+` operator (type mismatch)
- **Solution:** Convert the list to a string using `str([1, 2, 3])`, which produces `"[1, 2, 3]"`
- **Test cases verify:**
 1. The output equals the expected string with list representation
 2. Manual concatenation produces the same result
 3. The output is NOT the space-separated version (confirming `str()` conversion, not `join()`)

You can add this code to the end of your `ASS-7.5.py` file.

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:

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

The screenshot shows the Visual Studio Code interface with the following details:

- Editor:** The main editor window displays the code for `ASS-7.5.py`. It contains a function `sum_two_numbers` that reads two inputs from the user and returns their sum as a string. A test function `test_sum` is also present, which fails because it adds strings instead of integers.
- Terminal:** The terminal at the bottom shows the execution of the script. It prompts for two numbers, receives "5" and "3", and prints the result as "8". It then states "All tests passed ✓".
- CHAT:** A floating Chat window titled "FIXING STRING AND LIST CONCATENATION..." shows an assertion error: `assert combine() != "Numbers: 1 :"`.
- Explanation of the fix:** A section on the right provides an explanation of the bug and its fix. It notes that concatenating a string with a list using the + operator is a type mismatch. It suggests using `str([1, 2, 3])` to produce "[1, 2, 3]".
- Test cases verify:** It lists three points: 1. The output equals the expected string with list representation; 2. Manual concatenation produces the same result; 3. The output is NOT the space-separated version (confirming str() conversion, not join()).
- Additional note:** A message says you can add code to the end of the file.

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.

