

# ASSIGNMENT-7.5

HT.NO:2303A510H6

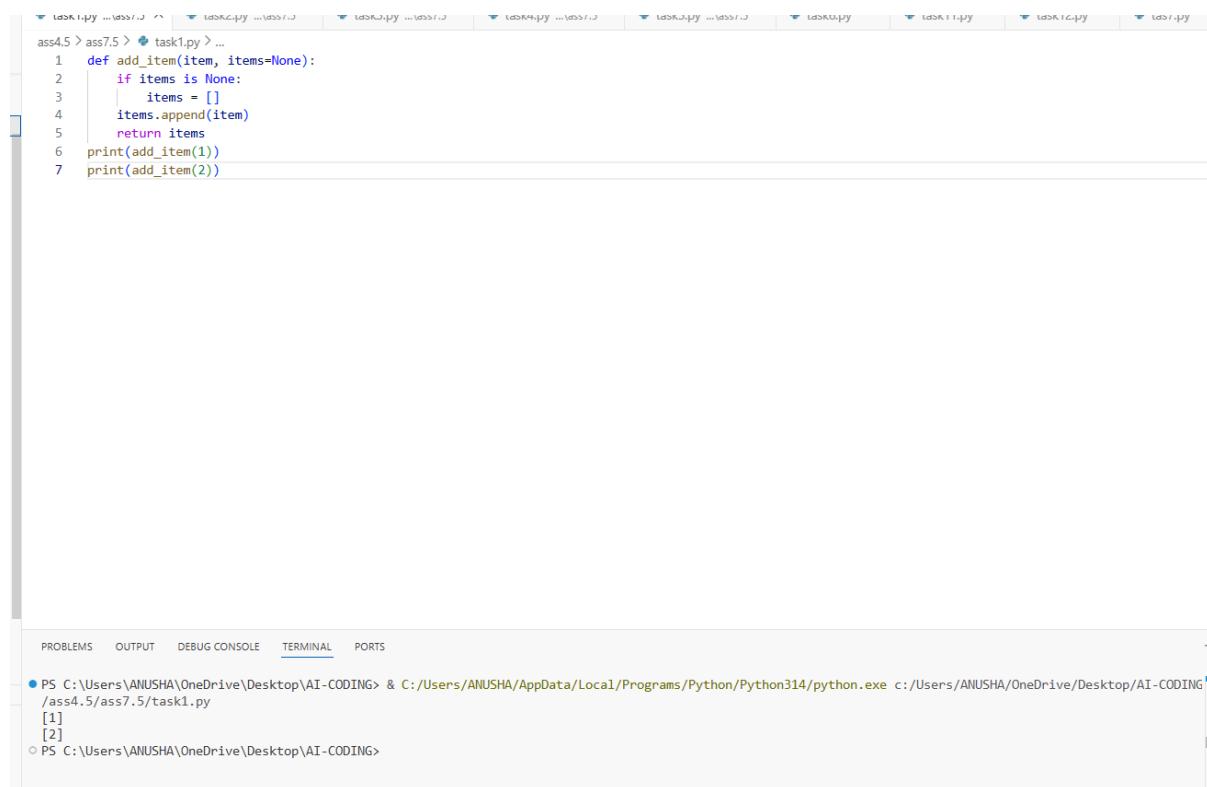
Batch No:30

## Task 1: Mutable Default Argument – Function Bug

### Prompt:

Analyze the given Python function where a mutable default argument causes unexpected behavior and generate a corrected version of the function.

### Code & Output:



The screenshot shows a code editor interface with multiple tabs at the top, all labeled "task1.py". Below the tabs is a code editor window containing the following Python code:

```
ass4.5 > ass7.5 > task1.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))
```

At the bottom of the screen is a terminal window showing the execution of the script and its output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING\ass4.5/ass7.5/task1.py
[1]
[2]
○ PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

### Observation:

The function correctly avoids the mutable default argument bug by creating a new list for each call, resulting in independent outputs for each item added.

## Task 2: Floating-Point Precision Error

### Prompt:

Analyze the given Python code where a floating-point comparison fails and generate a corrected version using tolerance.

### Code & Output:

```
ass4.5 > ass7.5 > task2.py > ...
1  def check_sum():
2      tolerance = 1e-10
3      return abs((0.1 + 0.2) - 0.3) < tolerance
4  print(check_sum())
```

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/ass4.5/ass7.5/
True
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

The terminal shows the execution of a Python script named task2.py. The script defines a function check\_sum() that calculates the absolute difference between the sum of 0.1 and 0.2 (which is 0.3) and 0.3 itself, then checks if this difference is less than a tolerance value of 1e-10. The output of the script is "True", indicating that the values are considered equal within the specified tolerance.

### Observation:

The function correctly handles floating-point precision issues by comparing values within a small tolerance instead of using direct equality.

## Task 3: Recursion Error – Missing Base Case

### Prompt:

Analyze the given recursive Python function that runs infinitely due to a missing base case and generate a corrected version with a proper stopping condition.

## Code & Output:

```
ass4.5 > ass7.5 > task3.py > ...
1  def countdown(n):
2      print(n)
3      if n == 0:
4          return
5      return countdown(n-1)
6  countdown(5)
```

The screenshot shows a code editor interface with a terminal window. The terminal tab is active, displaying the command PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/ass4.5/ass7.5/.task3.py. The output of the script is shown below, starting with 5 and ending at 0, which is the expected behavior for a correctly implemented recursive countdown function.

```
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/ass4.5/ass7.5/.task3.py
5
4
3
2
1
0
```

## Observation:

The recursion now stops correctly due to the added base case.  
Each function call reduces the value of n, preventing infinite recursion.  
The program executes safely and produces the expected countdown output.

## Task 4: Dictionary Key Error

### Prompt:

Fix the Key Error caused by accessing a missing dictionary key using .get() or error handling.

## Code & Output:

```
ass4.5 > ass7.5 > task4.py > ...
1  def get_value():
2      data = {"a": 1, "b": 2}
3      return data.get("c", "Key not found")
4  print(get_value())
```



A screenshot of a terminal window titled "Python". The window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The output pane shows the following text:

- PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/ass4.5/ass7.5/task4.py
- Key not found
- PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>

## Observation:

Since the key "c" is not available in the dictionary, the program prints "Key not found" instead of a number.

## Task 5: Infinite Loop – Wrong Condition

### Prompt:

Fix the infinite loop by updating the loop variable so the condition eventually becomes false.

## Code & Output:

```
ass4.5 > ass7.5 > task5.py > ...
1  def loop_example():
2      i = 0
3      while i < 5:
4          print(i)
5          i += 1
6  loop_example()
7
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + v ⌂ ⌂ ... ×
● PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/ass4.5/ass7.5/
task5.py
0
1
2
3
4
○ PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

## Observation:

The loop starts from 0 and prints the numbers 0, 1, 2, 3, and 4 because it runs while the value of *i* is less than 5. The value of *i* increases by 1 each time, so the loop stops automatically when *i* reaches 5.

## Task 6: Unpacking Error – Wrong Variables

### Prompt:

Fix the tuple unpacking error by matching the number of variables to the values or ignoring extra values using `_`.

### Code& Output:

```
ass4.5 > ass7.5 > task6.py > ...
1   a, b, c = (1, 2, 3)
2   print(a)
3   print(b)
4   print(c)
5
```

A screenshot of a terminal window titled "Python". The window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The terminal shows the command PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/ass4.5/ass7.5/task6.py followed by the output 1, 2, 3.

## Observation:

The values 1, 2, and 3 are assigned to variables a, b, and c in a single statement.  
The program prints 1, 2, and 3 on separate lines.

## Task 7: Mixed Indentation – Tabs vs Spaces

### Prompt:

Fix the Python code by correcting mixed indentation (tabs vs spaces) so it executes without errors.

### Code& Output:

```
ass4.5 > ass7.5 > tas7.py > ...
1  # Fix the Python code by correcting mixed indentation (tabs vs spaces) so it executes without errors
2  def func():
3      x = 5
4      y = 10
5      return x + y
6  result = func()
7  print(result)
```



The screenshot shows the VS Code interface with the 'TERMINAL' tab selected. The terminal window displays the following output:

```
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING> tas7.py
15
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

## Observation:

The program failed due to inconsistent indentation inside the function block. After applying consistent indentation using spaces, the code executed successfully and produced the correct output.

## Task 8: Import Error – Wrong Module Usage

### Prompt:

Fix the Python code by correcting the wrong module import so it runs without errors.

### Code&Output:

```
1 #Fix the Python code by correcting the wrong module import so it runs without errors
2 import math
3 print(math.sqrt(16))
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + □
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING
task8.py
4.0
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

## Observation:

The program produced an import error because an incorrect module name was used. After replacing it with the correct built-in “math” module, the code executed successfully and produced the expected output.

## Task 9: Unreachable Code – Return Inside Loop

### Prompt:

Fix the Python code by moving the return statement outside the loop so the function correctly calculates the total sum.

### Code& Output:

```
1  #Fix the Python code by moving the return statement outside the loop so the function correctly calculates the total sum
2  def total(numbers):
3      total_sum = 0
4      for n in numbers:
5          total_sum += n
6      return total_sum
7  print(total([1, 2, 3]))
```

A screenshot of a terminal window titled "Python". The window shows the following text:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + × ┌ └ ·
PS C:\Users\ANUASHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUASHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUASHA/OneDrive/Desktop/AI-CODING/ass4.py
6
PS C:\Users\ANUASHA\OneDrive\Desktop\AI-CODING>
```

## Observation:

The program returned a value during the first loop iteration due to a return statement inside the loop. After moving the return statement outside the loop and accumulating values properly, the function iterated through all elements and produced the correct total.

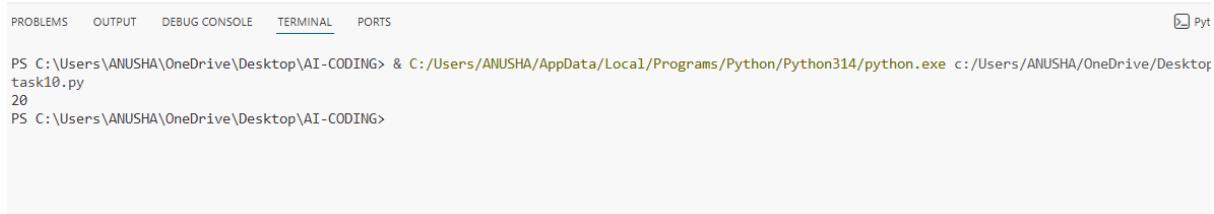
## Task 10: Name Error – Undefined Variable

### Prompt:

Analyze the given Python code where a variable is used before being defined, identify the error, and fix it by correcting the variable definition.

### Code&Output:

```
1 | 
2 def calculate_area(length, width):
3 |     return length * width
4 print(calculate_area(5, 4))
5
6
7
```



A screenshot of a terminal window from a code editor. The tabs at the top are PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and PORTS. On the right side, there is a small icon labeled 'Pyt'. The terminal output shows:

```
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/task10.py
20
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

## Observation:

The original code produced a `NameError` due to undefined variables. After passing `length` and `width` as parameters, the error was resolved and all assert test cases executed successfully.

## Task 11: Type Error – Mixing Data Types Incorrectly

### Prompt:

Analyze the given Python code that performs addition using type conversion, explain how the operation works, and verify that the program executes without errors.

### Code& Output:

```
ass4.5 > ass7.5 > task11.py > ...
1
2 def add_values():
3     return 5 + int("10")
4 print(add_values())
5
```

## **Observation:**

The code executes successfully without any error. The string value "10" is converted into an integer using int(), allowing valid addition with the integer 5, and the final output is 15.

## Task 12: Type Error – String + List Concatenation

## Prompt:

Analyze the Python code that converts a list to a string for concatenation, explain the behavior, and observe the output.

## Code&Output:

```
ass4.5 > ass7.5 > task12.py > ...
1 | def combine():
2 |     return "Numbers: " + str([1, 2, 3])
3 | print(combine())
4 |
```

The screenshot shows a terminal window with the following interface elements:

- Top bar: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), PORTS.
- Bottom right corner: Python icon, a '+' sign, and a close button.

The terminal output is:

- PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/task12.py
- Numbers: [1, 2, 3]
- PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>

## Observation:

The code executes successfully without any errors. The list [1, 2, 3] is converted into a string using `str()`, allowing it to be concatenated with the string "Numbers: ". The output displayed is Numbers: [1, 2, 3].

## Task 13: Type Error – Multiplying String by Float

### Prompt:

Analyze the Python code where a string is multiplied by a float, explain the type error, fix it by converting the float to an integer, and add three assert-based test cases.

### Code& Output:

The screenshot shows a code editor interface with a Python file open. The code defines a function `repeat_text` that multiplies the string "Hello" by 2.5 and prints the result. It also includes three test cases using `assert` statements to check if the output is "HelloHello", has a length of 10, and starts with "Hello". Below the code editor is a terminal window showing the command `python task13.py` being run, which outputs "HelloHello".

```
1 # Analyze the Python code where a string is multiplied by a float, explain the type error, fix it by converting the float to an integer, and add three a
2 def repeat_text():
3     return "Hello" * int(2.5)
4
5 print(repeat_text())
6 # Test cases
7 assert repeat_text() == "HelloHello"
8 assert len(repeat_text()) == 10
9 assert repeat_text().startswith("Hello")
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ⌂ ⌂ ...
```

```
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/ass4.5/a
task13.py
HelloHello
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

## Observation:

The program raised a `TypeError` because Python does not allow multiplication of a string by a float. Strings can only be multiplied by integers to indicate repetition. After converting the float value to an integer, the code executed successfully and produced the expected output.

## Task 14: Type Error – Adding None to Integer

### Prompt:

Analyze the Python code where `None` is added to an integer, identify the `TypeError`, explain why `NoneType` cannot be used in arithmetic operations, fix the issue by assigning a default value, and validate the fix using `assert` statements.

### Code& Output:

The screenshot shows a code editor window with a Python script named task14.py. The code defines a function compute() that adds 10 to a value. It handles the case where the value is None by assigning it a default value of 0. The code includes several assert statements to test the function's behavior. Below the code editor is a terminal window showing the execution of the script and its output.

```
ass4.5 > ass7.5 > task14.py > ...
1 #Analyze the Python code where None is added to an integer, identify the TypeError, explain why NoneType cannot be used in arithmetic operations, fix thi
2 def compute():
3     value = None
4     if value is None:
5         value = 0 # Assign a default value to avoid TypeError
6     return value + 10
7
8 print(compute())
9 # Test cases
10 assert compute() == 10
11 assert isinstance(compute(), int)
12 assert compute() >= 0
```

TERMINAL

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

```
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/ass4
task14.py
10
PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>
```

## Observation:

The program raised a `TypeError` because `NoneType` does not support arithmetic operations with integers. Since `None` represents the absence of a value, it cannot be added to a number. By assigning a valid default integer value, the computation executed successfully and produced the correct output.

## Task 15: Type Error – Input Treated as String Instead of Number

### Prompt:

Fix the code by converting user input from string to integer using `int()` so the numbers are added correctly instead of concatenated.

### Code&Output:

```
ass4.5 > ass7.5 > task15.py > ...
1 def sum_two_numbers():
2     a = int(input("Enter first number: "))
3     b = int(input("Enter second number: "))
4     return a + b
5 print(sum_two_numbers())
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

● PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING> & C:/Users/ANUSHA/AppData/Local/Programs/Python/Python314/python.exe c:/Users/ANUSHA/OneDrive/Desktop/AI-CODING/task15.py  
Enter first number: 55  
Enter second number: 89  
144  
○ PS C:\Users\ANUSHA\OneDrive\Desktop\AI-CODING>

## Observation:

The program asks the user to enter two numbers and changes them into whole numbers. Then it adds those two numbers together and shows the total.