

## **Assignment-7.5**

**2303A51062**

**Batch:29**

### **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.

## Code:

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

- Editor:** Untitled-1.py (file path: C:\Users\DELL\OneDrive\Documents\Untitled-1.py)
- Code Content:**

```
1 class Car:
2     wheels = 4
3
4     def __init__(self, color):
5         self.color = color
6
7
8 rolls_royce = Car('blue')
9 bmw = Car('black')
10
11 print(f'{car.wheels=}')
12 print(f'{rolls_royce.wheels=}')
13 print(f'{bmw.wheels=}')
14
```
- Terminal Output:**

```
Data\Local\Programs\Microsoft VS Code & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe c:/Users/DELL/OneDrive/Documents/Untitled-1.py
False
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe c:/Users/DELL/OneDrive/documents/Untitled-1.py
True
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe c:/Users/DELL/OneDrive/Documents/Untitled-1.py
Car.wheels=4
rolls_royce.wheels=4
bmw.wheels=4
```

## Justification:

`wheels` is a class variable, so it belongs to the class `Car` and is shared by all its objects.

Accessing `wheels` using `Car`, `rolls_royce`, or `bmw` gives the same value because Python looks for the attribute in the instance first and, if not found, in the class.

## 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

**Code:**

The screenshot shows a code editor in VS Code with two tabs: '#1 (AI-Based Code Completion for Conditio.py)' and 'Untitled-1.py'. The 'Untitled-1.py' tab contains the following code:

```
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 def check_sum():
2     return abs((0.1 + 0.2) - 0.3) < 1e-9
3
4 print(check_sum())
5
```

Below the code editor, the terminal window shows the execution of the script:

```
PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

Data\Local\Programs\Microsoft VS Code & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe c:/Users/DELL/OneDri
ve/Documents/Untitled-1.py
False
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe
c:/Users/DELL/OneDrive/Documents/Untitled-1.py
True
○ PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code>
```

### **Justification:**

Floating-point numbers are stored in binary, so  $0.1 + 0.2$  does not equal exactly  $0.3$ . Using a small tolerance (epsilon) allows comparison within an acceptable error range, making the check reliable.

### **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 :**

## Code:

The screenshot shows a Microsoft VS Code interface. At the top, there are two tabs: '#1 (AI-Based Code Completion for Conditio.py)' and 'Untitled-1.py'. The 'Untitled-1.py' tab is active, showing the following Python code:

```
1 def countdown(n):
2     if n == 0:
3         print("Blast off!")
4         return
5     print(n)
6     return countdown(n-1)
7 countdown(5)
8
```

Below the code editor, there are several tabs: PROBLEMS (8), OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, displaying the output of the Python script:

```
● PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe c:/Users/DELL/OneDrive/Documents/Untitled-1.py
5
4
3
2
1
Blast off!
```

The terminal also shows a prompt: '○ PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code>'.

## Justification

The original function had **no base case**, so it kept calling itself endlessly and caused a recursion error.

Adding a base case (`n == 0`) gives the function a clear stopping condition, preventing infinite recursion.

## 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.

**Code:**

The screenshot shows the VS Code interface with two tabs open: '#1 (AI-Based Code Completion for Conditi.py)' and 'Untitled-1.py'. The code in 'Untitled-1.py' is as follows:

```
C:\> Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 # Bug: Accessing non-existing key
2 def get_value():
3     data = {"a": 1, "b": 2}
4     return data["c", "Key not found"]
5
6 print(get_value())
7
8
```

The terminal below shows the execution of the code. It highlights the line `return data["c"]` with a red squiggle under the closing brace, indicating a syntax error. The output shows the error message and the resulting KeyError:

```
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe
c:/Users/DELL/OneDrive/Documents/Untitled-1.py
    print(get_value())
        ~~~~~^
File "c:/Users/DELL/OneDrive/Documents/Untitled-1.py", line 4, in get_value
    return data["c"]
        ~~~~~^
KeyError: 'c'
● PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe
c:/Users/DELL/OneDrive/Documents/Untitled-1.py
Key not found
```

## **Justification:**

Accessing a missing key with `data[ "c" ]` raises a `KeyError`.

Using `dict.get()` returns a default value instead, preventing the runtime error.

## **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

**code::**

The screenshot shows a Microsoft VS Code interface. At the top, there are two tabs: '#1 (AI-Based Code Completion for Conditi.py)' and 'Untitled-1.py X'. The file path 'C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...' is displayed above the code editor. The code editor contains the following Python script:

```
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 def loop_example():
2     i = 0
3     while i < 5:
4         print(i)
5         i += 1    # increment to avoid infinite loop
6
7 loop_example()
8
```

Below the code editor is a terminal window showing the execution of the script:

```
PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\DELL\OneDrive\Documents\Untitled-1.py
● PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe
c:/Users/DELL/OneDrive/Documents/Untitled-1.py
Key not found
● PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe
c:/Users/DELL/OneDrive/Documents/Untitled-1.py
0
1
2
3
4
```

### Justification:

In the original code, `i` was never updated, so the condition `i < 5` was always true. Incrementing `i` inside the loop ensures the condition eventually becomes false, stopping the loop.

### 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.**

**Code:**

```
#1 (AI-Based Code Completion for Conditionals)
C:\Users\DELL\OneDrive\Documents\Untitled-1.py
1 a, b, _ = (1, 2, 3)
2 print(a, b)
3
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe
c:/Users/DELL/OneDrive/Documents/Untitled-1.py
2
3
4
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe
c:/Users/DELL/OneDrive/Documents/Untitled-1.py
1 2
```

## Justification

Tuple unpacking requires the **number of variables to match the number of values**. Using `_` allows you to intentionally ignore extra values without causing an error.

## 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.

**code:**

The screenshot shows a Microsoft VS Code interface. The top bar displays the title 'Untitled-1.py'. The code editor contains the following Python code:

```
1 def func():
2     x = 5
3     y = 10
4     return x + y
5
6 print(func())
7
```

The terminal at the bottom shows the following output:

```
PS C:\Users\DELL\OneDrive\Documents\Untitled-1.py
PS C:\Users\DELL\OneDrive\Documents\Untitled-1.py
PS C:\Users\DELL\OneDrive\Documents\Untitled-1.py
PS C:\Users\DELL\OneDrive\Documents\Untitled-1.py
```

The terminal output indicates multiple errors, likely due to the inconsistent indentation causing syntax errors.

### **Justification:**

Mixed indentation causes an **IndentationError** because Python requires consistent use of spaces or tabs within the same block. Aligning all statements at the same indentation level using spaces fixes the execution error.

### **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

### **Code:**

The screenshot shows a dark-themed interface of the Visual Studio Code editor. At the top, there are two tabs: '#1 (AI-Based Code Completion for Conditipy' and 'Untitled-1.py'. The 'Untitled-1.py' tab is active. Below the tabs, the code editor displays the following Python code:

```
C:\> Users > DELL > OneDrive > Documents > Untitled-1.py
1 import maths
2 print(math.sqrt(16))
3
```

At the bottom of the screen, the terminal window shows the command line path and the command run:

```
PS C:\Users\DELL\OneDrive\Documents> & c:/users/DELL/oneDrive/Documents/Untitled-1.py
4.0
```

The terminal also shows some status icons and text related to powershell and python.exe.

## **Justification:**

The module name is `math`, not `maths`. Importing the correct built-in module fixes the `ImportError` and allows access to functions like `sqrt()`.

## **Task 9 (Unreachable Code – Return Inside Loop)**

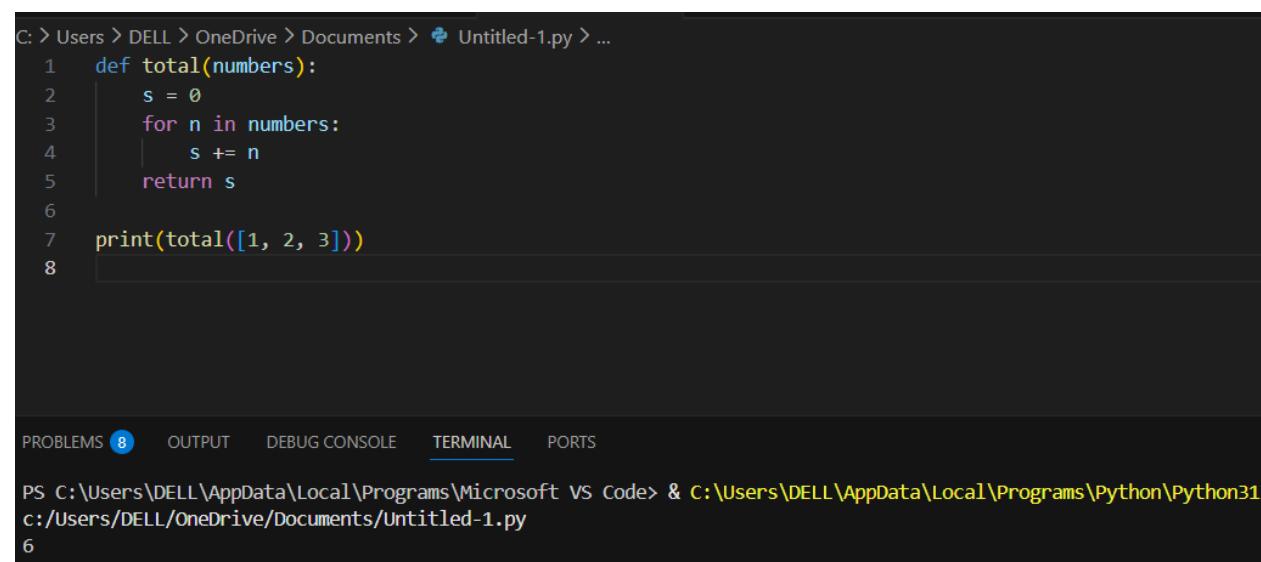
**Task:** Analyze given code where a return inside a loop prevents full iteration. Use AI to fix it.

# 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.

### **Code:**



C:\> Users > DELL > OneDrive > Documents > Untitled-1.py > ...

```
1 def total(numbers):
2     s = 0
3     for n in numbers:
4         s += n
5     return s
6
7 print(total([1, 2, 3]))
```

PROBLEMS 8    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & c:\Users\DELL\AppData\Local\Programs\Python\Python31  
c:/Users/DELL/OneDrive/Documents/Untitled-1.py

### **Justification:**

The `return` statement inside the loop causes the function to exit after the first iteration. Moving the `return` outside the loop allows all elements to be processed and the total sum to be computed correctly

### **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.

**code:**

The screenshot shows a Python script named Untitled-1.py in a code editor. The code defines a function calculate\_area that takes length and width as parameters and returns their product. It includes three test cases using assert statements to check the function's output against expected values. A final print statement outputs "All tests passed". Below the code editor, the terminal tab is active, showing the command PS C:\Users\DELL\OneDrive\Documents\Untitled-1.py and the output "All tests passed".

```
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 def calculate_area(length, width):
2     return length * width
3
4 # Test cases
5 assert calculate_area(5, 4) == 20
6 assert calculate_area(10, 2) == 20
7 assert calculate_area(7, 3) == 21
8
9 print("All tests passed")
10
11

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\DELL\OneDrive\Documents\Untitled-1.py
c:/Users/DELL/OneDrive/Documents/Untitled-1.py
All tests passed
```

**Justification:**

The error occurs because `length` and `width` are referenced before being defined. Defining them as function parameters ensures they are provided when the function is called, preventing the `NameError` and making the function logically correct and reusable.

**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.

### code:

The screenshot shows a Python script named 'Untitled-1.py' in a code editor. The code defines a function 'add\_values' that returns the sum of two integers. It includes three test cases using the 'assert' keyword to check if the function's output matches expected values. The code editor interface includes tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. Below the editor, the terminal window displays the command 'python Untitled-1.py' and the output 'All tests passed' repeated twice.

```
# 1 (AI-Based Code Completion for Conda) Untitled-1.py ...
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 def add_values():
2     return 5 + int("10")
3
4 # Test cases
5 assert add_values() == 15
6 assert 3 + int("7") == 10
7 assert 0 + int("20") == 20
8
9 print("All tests passed")
10
```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe c:/Users/DELL/OneDrive/Documents/Untitled-1.py

All tests passed

All tests passed

### Justification:

Python does not support adding an integer and a string directly because they are different data types. Converting the string to an integer ensures both operands are compatible, preventing the `TypeError` and allowing correct arithmetic.