

## 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
```

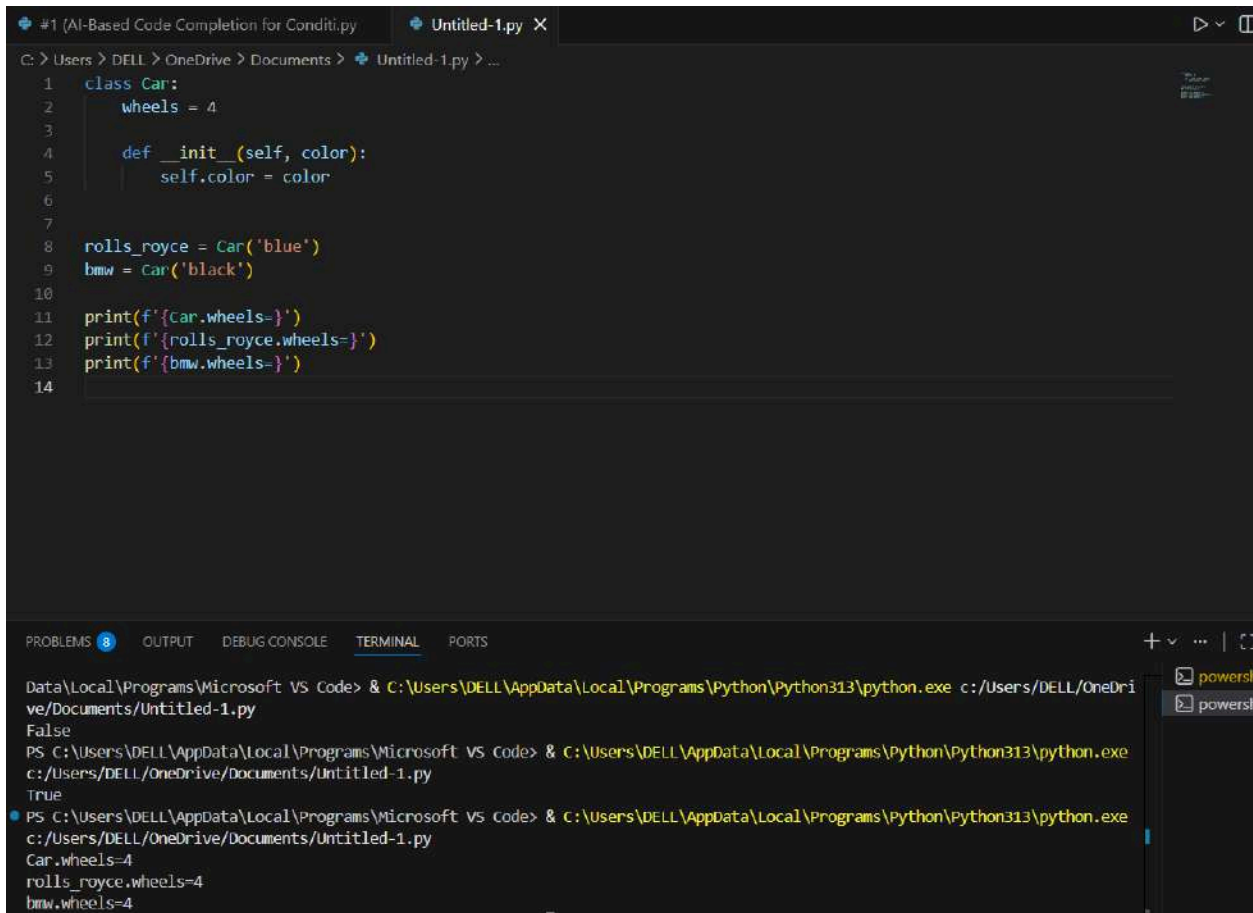
```
print(add_item(1))
```

```
print(add_item(2))
```

#### **Expected Output:**

Corrected function avoids shared list bug.

## Code:



```
#1 (AI-Based Code Completion for Conditio.py)  Untitled-1.py X
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...

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

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/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)

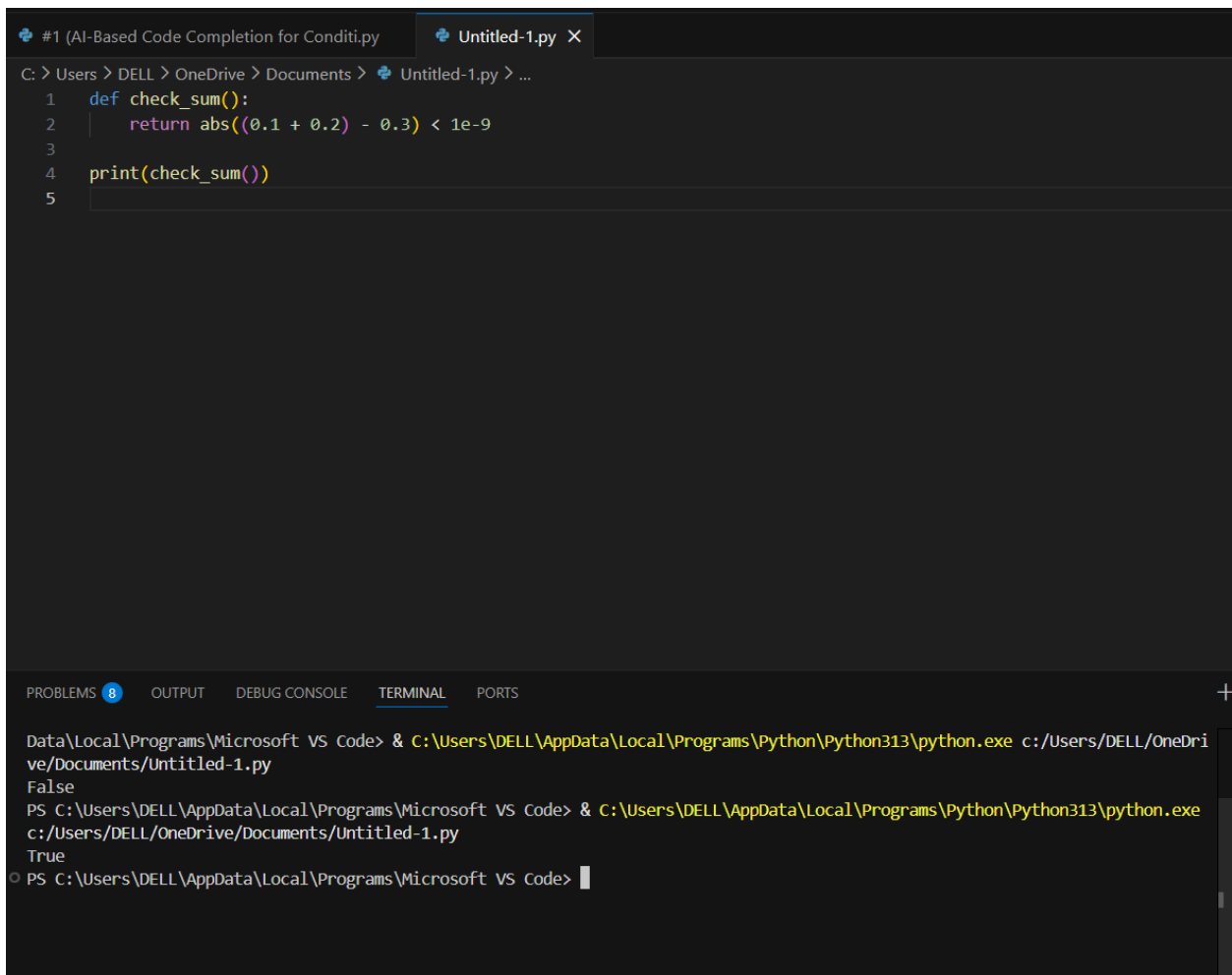
**Task:** Analyze given code where floating-point comparison fails. Use AI to correct with tolerance.

## # Bug: Floating point precision issue

```
def check_sum():  
  
    return (0.1 + 0.2) == 0.3  
  
print(check_sum())
```

**Expected Output:** Corrected function

**Code:**



```
#1 (AI-Based Code Completion for Conditi.py)  Untitled-1.py X  
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  
  
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/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> |
```

**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)**

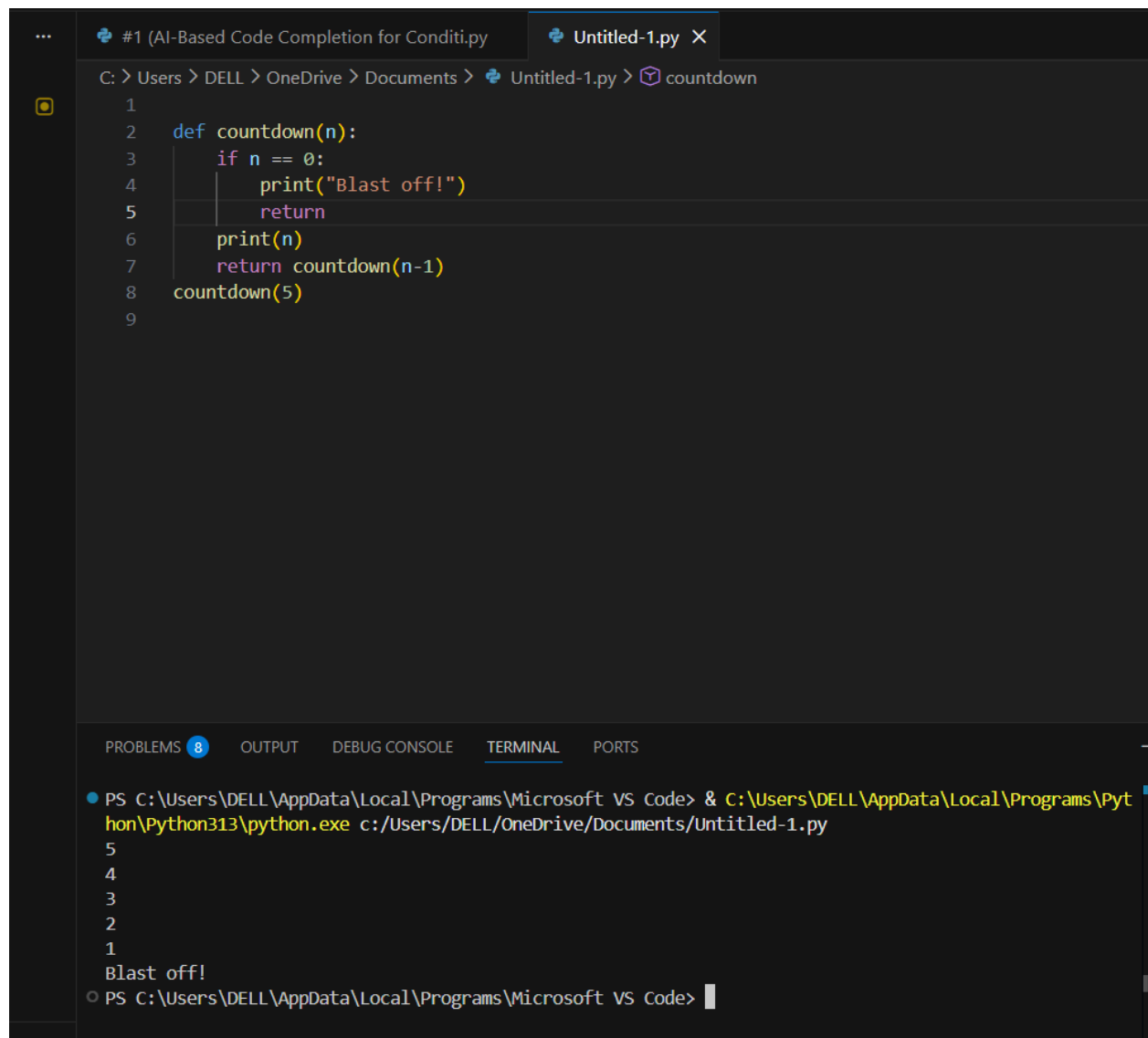
**Task:** Analyze given code where recursion runs infinitely due to missing base case. Use AI to fix.

**# Bug: No base case**

```
def countdown(n):  
  
    print(n)  
  
    return countdown(n-1)  
  
countdown(5)
```

**Expected Output :**

## Code:



```
#1 (AI-Based Code Completion for Condi.py)  Untitled-1.py X
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > countdown
1
2 def countdown(n):
3     if n == 0:
4         print("Blast off!")
5         return
6     print(n)
7     return countdown(n-1)
8 countdown(5)
9

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
5
4
3
2
1
Blast off!
○ 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)

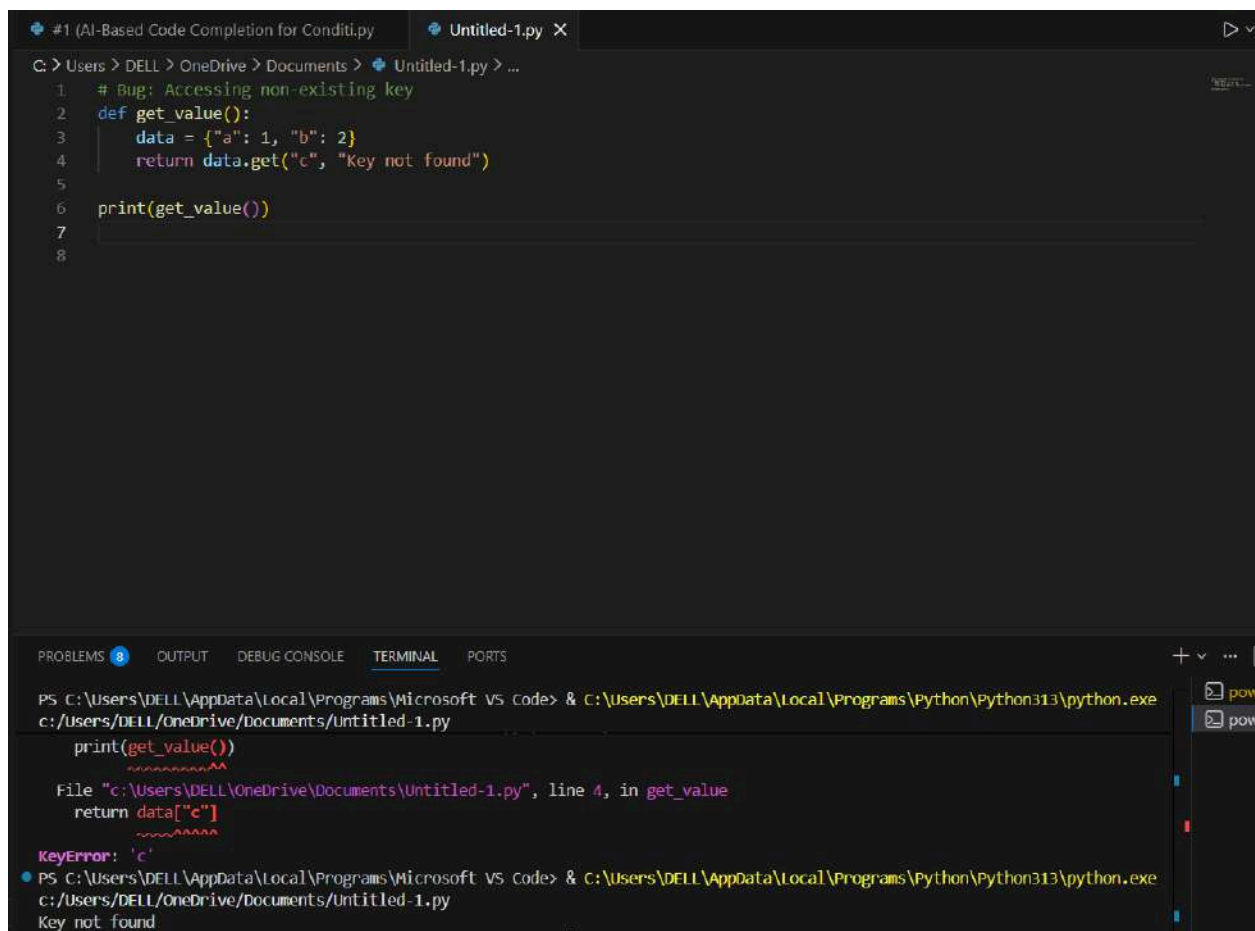
Task: Analyze given code where a missing dictionary key causes error. Use AI to fix it.

### # 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 a VS Code editor window with a file named 'Untitled-1.py'. The code in the editor is as follows:

```
1 # Bug: Accessing non-existing key  
2 def get_value():  
3     data = {"a": 1, "b": 2}  
4     return data.get("c", "key not found")  
5  
6 print(get_value())  
7  
8
```

The bottom panel of the VS Code interface shows the 'TERMINAL' tab. It displays the command prompt output for running the 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  
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:.**

```
#1 (AI-Based Code Completion for Conditio.py)  Untitled-1.py X
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

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
• 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)

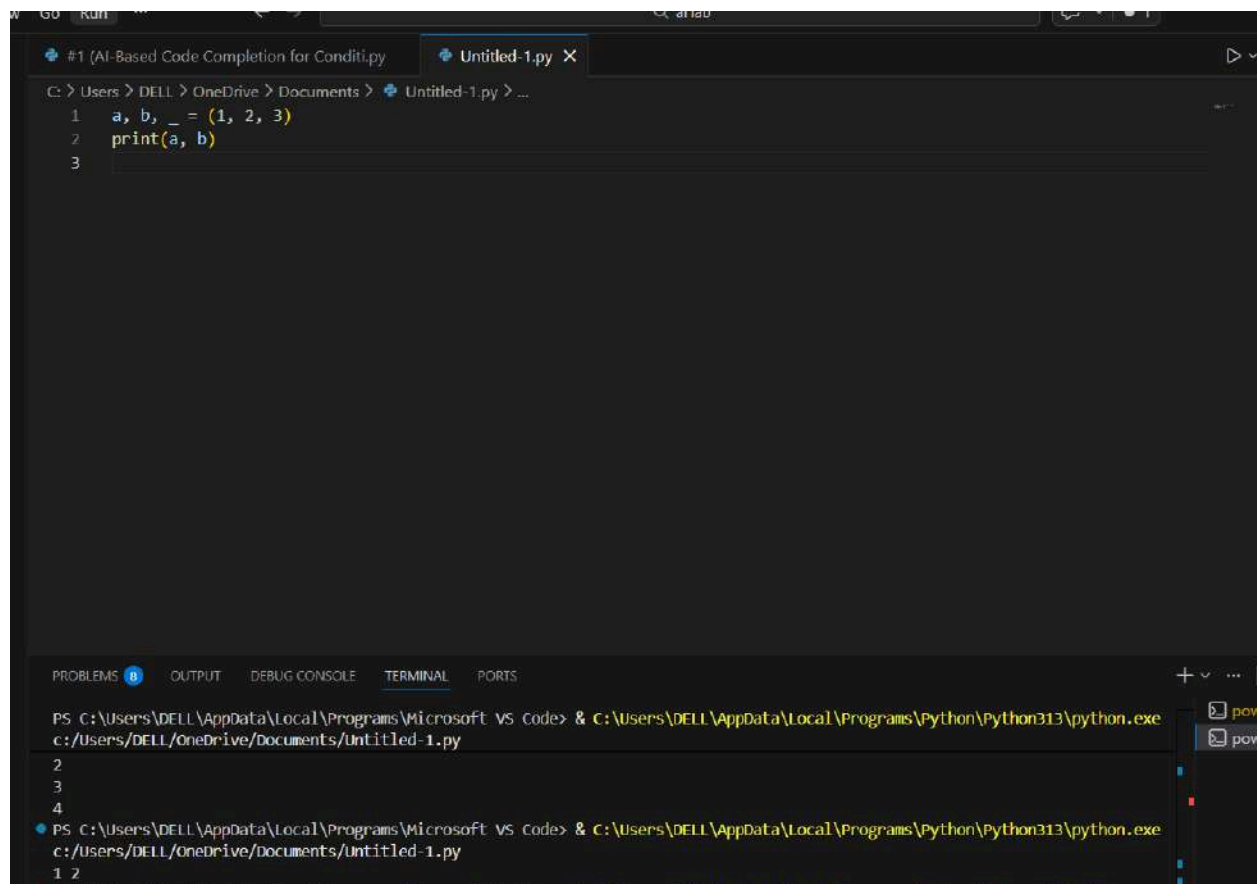
Task: Analyze given code where tuple unpacking fails. Use AI to fix it.

## # Bug: Wrong unpacking

a, b = (1, 2, 3)

**Expected Output: Correct unpacking or using \_ for extra values.**

**Code:**



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

PROBLEMS 0 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():
```

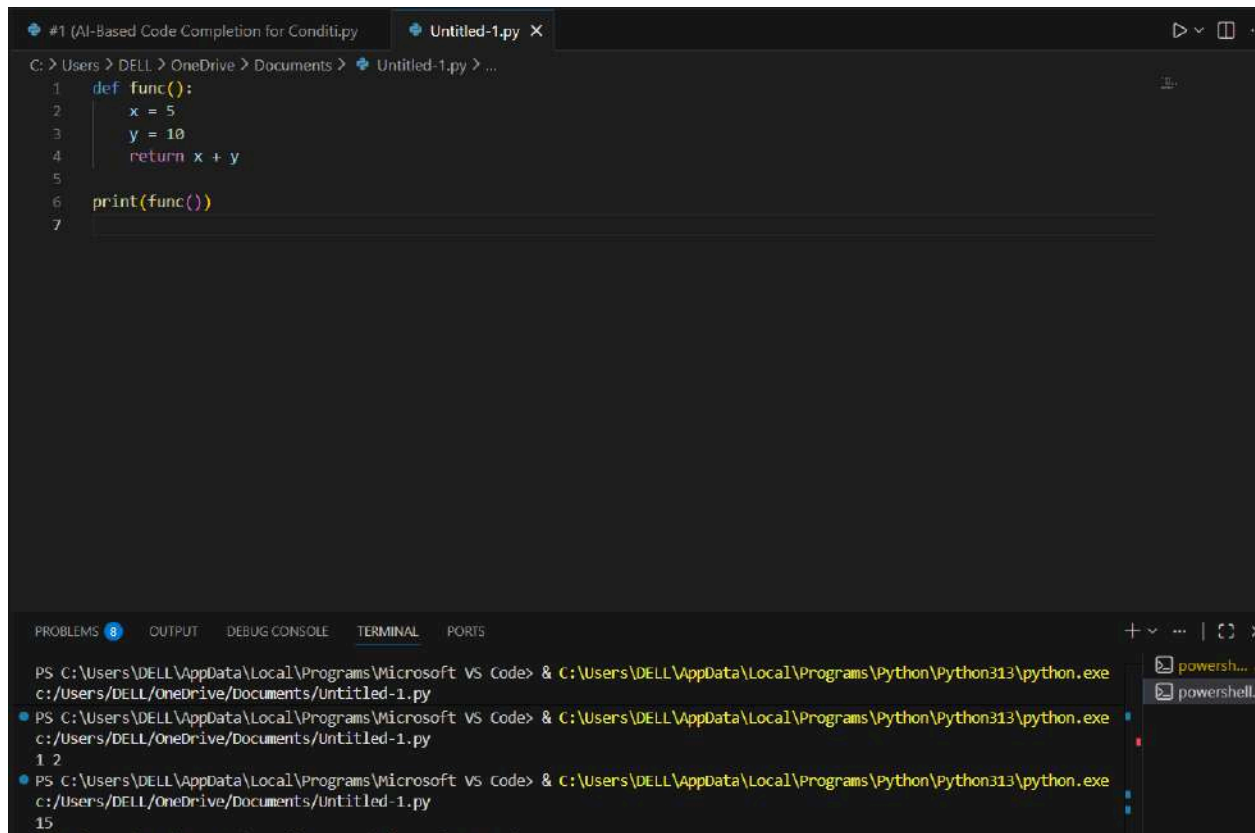
```
    x = 5
```

```
        y = 10
```

```
    return x+y
```

**Expected Output :** Consistent indentation applied.

**code:**



The screenshot shows a code editor with a file named 'Untitled-1.py'. The code in the editor is as follows:

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

Below the code editor is a terminal window. It shows the command prompt running the Python script. The output of the script is 15, which is the sum of 5 and 10. The terminal output is as follows:

```
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
15
```

## 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.

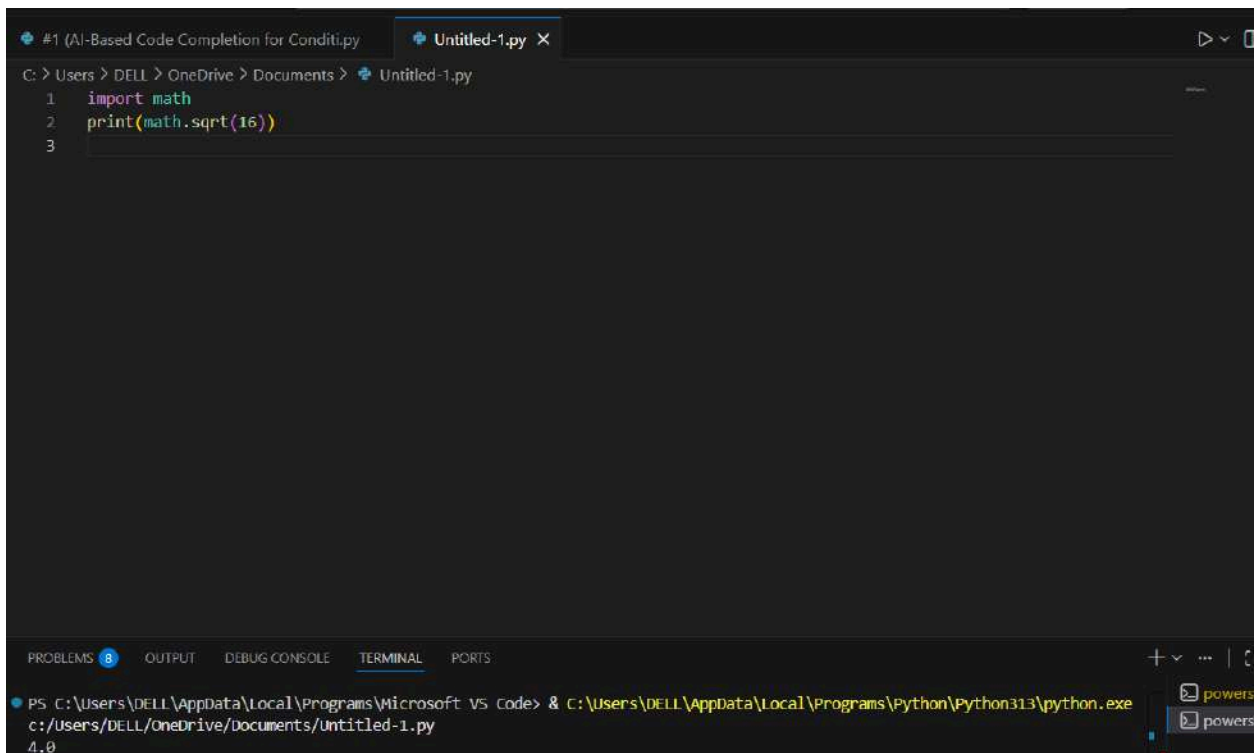
# Bug: Wrong import

```
import maths
```

```
print(maths.sqrt(16))
```

**Expected Output:** Corrected to import math

**Code:**



```
#1 (AI-Based Code Completion for Conditio.py)  Untitled-1.py X
C: > Users > DELL > OneDrive > Documents > Untitled-1.py
1  import math
2  print(math.sqrt(16))
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
4.0
```

## 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]))
8

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code> & C:\Users\DELL\AppData\Local\Programs\Python\Python311\python.exe c:/Users/DELL/OneDrive/Documents/Untitled-1.py
6
```

**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:**

```
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\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

## 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)`).

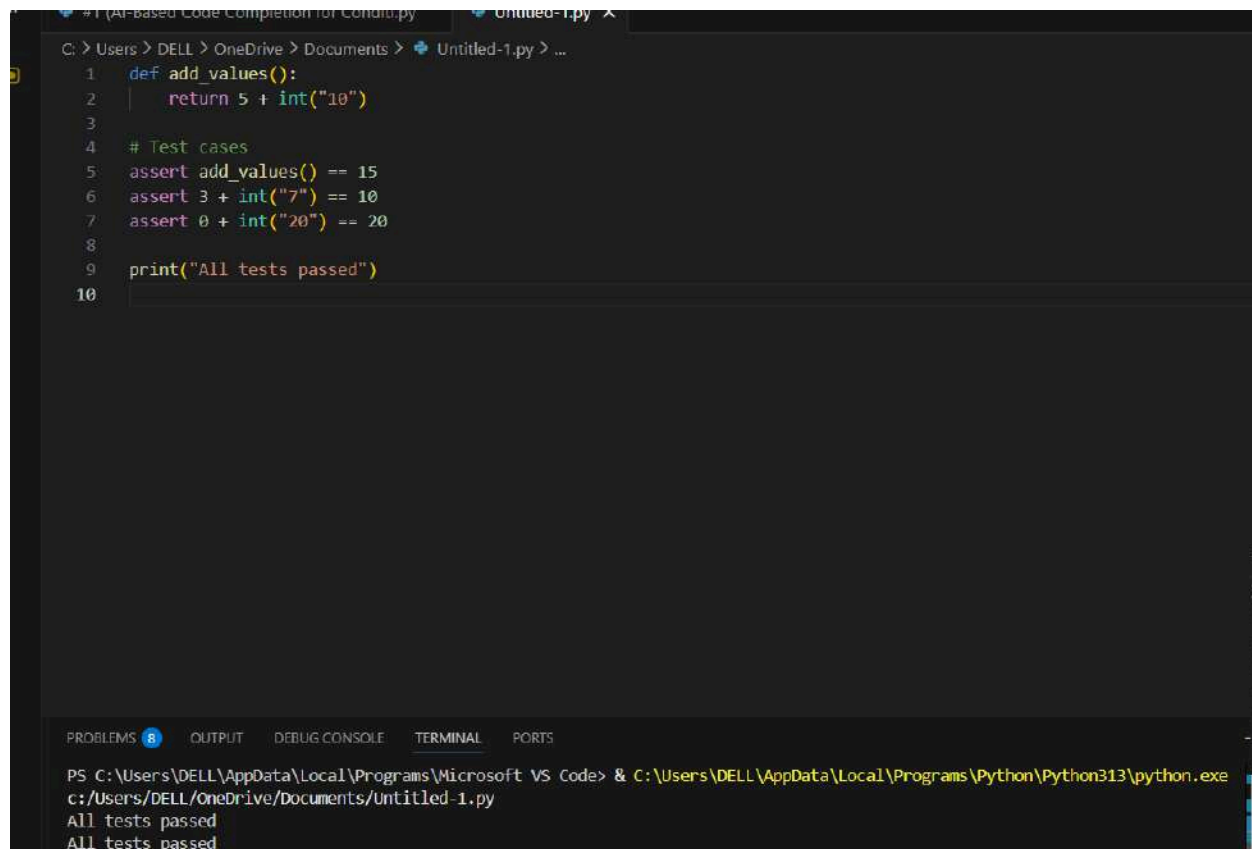
- Verify with 3 assert cases.

### Expected Output #6:

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

Successful test validation.

**code:**



```
# 1 (AI-based Code Completion for Condiu.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 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.