

## **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 Microsoft Visual Studio Code interface. At the top, there are two tabs: '#1 (AI-Based Code Completion for Conditi.py)' and 'Untitled-1.py'. The code editor contains the following Python script:

```
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
```

Below the code editor, the terminal window displays 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/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 the Microsoft VS Code interface. On the left, there's a code editor window titled "Untitled-1.py" containing the following Python code:

```
def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-9
print(check_sum())
```

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

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

The terminal window has tabs at the bottom labeled "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "TERMINAL", and "PORTS". The "TERMINAL" tab is currently selected.

## **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 code editor window in VS Code. The top bar has tabs for '#1 (AI-Based Code Completion for Conditi.py)' and 'Untitled-1.py'. The Untitled-1.py tab is active, showing the following Python code:

```
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
```

Below the code editor is a navigation bar with tabs: PROBLEMS (8), OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is selected, displaying the output of 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
- 5
- 4
- 3
- 2
- 1
- Blast off!

## 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 a Python file named 'Untitled-1.py' open. The code contains a bug where it tries to access a non-existent key 'c' in a dictionary. The terminal below shows the execution of the script and the resulting error message.

```
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.get("c", "Key not found")
5:
6: print(get_value())
7:
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
  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 the VS Code interface with the following details:

- Editor:** The code editor has two tabs: "#1 (AI-Based Code Completion for Conditipy)" and "Untitled-1.py". The code in "Untitled-1.py" is:

```
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
```
- Terminal:** The terminal window at the bottom shows the command line and its output:

```
c:\Users\DELL\OneDrive\Documents> python Untitled-1.py
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:**

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 Conditio.py)' and 'Untitled-1.py'. The code editor displays the following Python code:

```
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 a, b, _ = (1, 2, 3)
2 print(a, b)
3
```

Below the code editor is a 'PROBLEMS' tab with a count of 8, which is highlighted. The 'OUTPUT' tab is also visible. The terminal window at the bottom shows the command line path and the execution of 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
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 Python script named 'Untitled-1.py' in the VS Code editor. The code contains the following:

```
def func():
    x = 5
    y = 10
    return x + y
print(func())
```

The code uses tabs for indentation. The first two lines ('def func()' and 'x = 5') are indented with one tab. The third line ('y = 10') is indented with two tabs. The fourth line ('return x + y') is indented with one tab. The fifth line ('print(func())') is not indented. This inconsistency in indentation is highlighted by the question.

In the bottom right corner of the terminal, there are two PowerShell entries:

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

Both entries show the output '1 2'.

## **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 instance of VS Code. In the top left, there's a tab labeled '#1 (AI-Based Code Completion for Condit.py)' and another labeled 'Untitled-1.py'. The main editor area contains the following code:

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

Below the editor is a terminal window showing the command to run the script:

```
PS C:\Users\DELL\OneDrive\Documents> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe Untitled-1.py
```

The bottom navigation bar includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is currently selected. The status bar at the bottom right shows the Python version as 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:

The screenshot shows a code editor window with Python code. The code defines a function `total` that sums up a list of numbers. However, it contains an early `return` statement inside the loop, which causes it to only return the first element of the list. Below the code, the terminal shows the command `print(total([1, 2, 3]))` and its output, which is the number 1, indicating the bug.

```
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\Python31
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:

The screenshot shows a terminal window in VS Code with the following 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
```

Below the code, the terminal output shows:

```
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:

The screenshot shows a code editor in VS Code with a dark theme. The file is named 'Untitled-1.py'. The code defines a function 'add\_values' that returns the sum of an integer and a string converted to an integer. It includes three test cases using 'assert' statements and a final print statement. Below the code editor is a terminal window showing the command 'python Untitled-1.py' being run, followed by the output 'All tests passed' appearing twice.

```

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.

## **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

## Code:

The screenshot shows a Microsoft VS Code interface. At the top, there's a status bar with 'Untitled-1.py' and a progress bar. Below it is a code editor with the following Python script:

```
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 def combine():
2     return "Numbers: " + str([1, 2, 3])
3
4 # Test cases
5 assert combine() == "Numbers: [1, 2, 3]"
6 assert "Numbers: " + str([4, 5]) == "Numbers: [4, 5]"
7 assert "List: " + str([]) == "List: []"
8
9 print("All tests passed")
10
```

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

```
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
○ PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code>
```

## Justification:

Type conversion ensures compatibility between operands. Converting the list to a string prevents the **TypeError** and allows correct string concatenation, while assertions confirm the fix works as expected.

## 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.

## Code:

The screenshot shows a Microsoft VS Code interface. In the center, there's a terminal window displaying Python code and its execution results. The code defines a function `repeat_text` that multiplies a string by a float value converted to an integer. It includes three assert statements to verify the function's correctness. The terminal output shows the code was run in a PowerShell environment, and it successfully passed all three test cases.

```
C: > Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 def repeat_text():
2     return "Hello" * int(2.5)
3
4 # Test cases
5 assert repeat_text() == "HelloHello"
6 assert "Hi" * int(3.7) == "HiHiHi"
7 assert "A" * int(1.9) == "A"
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
○ PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code>
```

## Justification:

Converting the float to an integer ensures a valid repetition count, prevents the `TypeError`, and allows the code to execute correctly with predictable results.

## Task 14 (Type Error – Adding None to Integer)

Task: Analyze code where `None` is added to an integer.

### # Bug: Adding None and integer

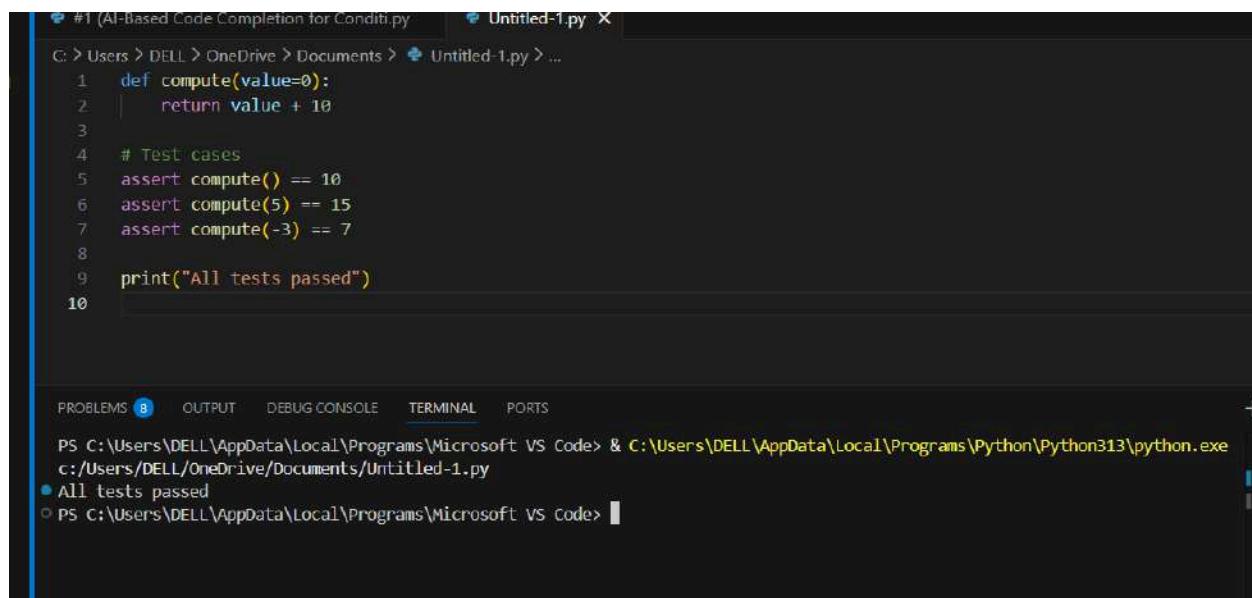
```
def compute():
```

```
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.

### code:



The screenshot shows a VS Code interface with a dark theme. In the center, there's a code editor window titled "Untitled-1.py" containing the following Python code:

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

Below the code editor is a terminal window showing the command line path and the execution of the script:

```
PS C:\Users\DELL\OneDrive\Documents> & C:\Users\DELL\AppData\Local\Programs\Python\Python313\python.exe c:/Users/DELL/OneDrive/Documents/Untitled-1.py  
● All tests passed  
○ PS C:\Users\DELL\OneDrive\Documents>
```

### Justification:

Assigning a default numeric value ensures valid arithmetic, prevents the `TypeError`, and makes the function safe and predictable.

## 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: ")

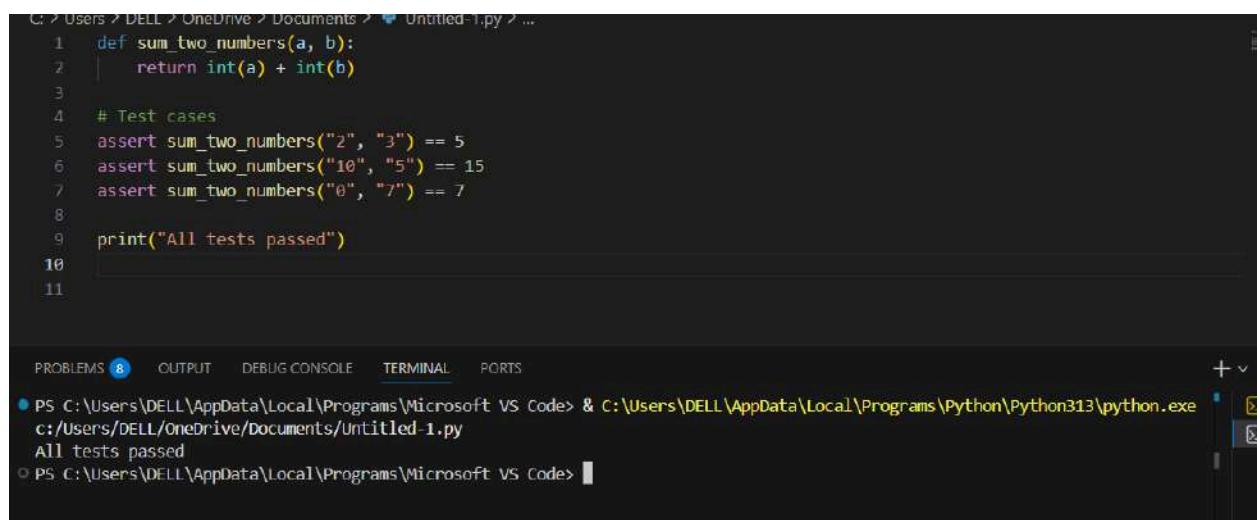
    return a + b

print(sum_two_numbers())
```

### Requirements:

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

### code:



The screenshot shows a Microsoft VS Code interface. The code editor contains the following Python script:

```
C:\> Users > DELL > OneDrive > Documents > Untitled-1.py > ...
1 def sum_two_numbers(a, b):
2     return int(a) + int(b)
3
4 # Test cases
5 assert sum_two_numbers("2", "3") == 5
6 assert sum_two_numbers("10", "5") == 15
7 assert sum_two_numbers("0", "7") == 7
8
9 print("All tests passed")
10
11
```

The terminal tab at the bottom shows the following output:

```
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
PS C:\Users\DELL\AppData\Local\Programs\Microsoft VS Code>
```

**Justification:**

Type conversion changes string inputs into numbers, preventing logical errors and enabling correct arithmetic. Assertions confirm the fix works correctly for multiple cases.