AI ASSISTED -CODING

ASSIGNMENT 7.5

2303A51918

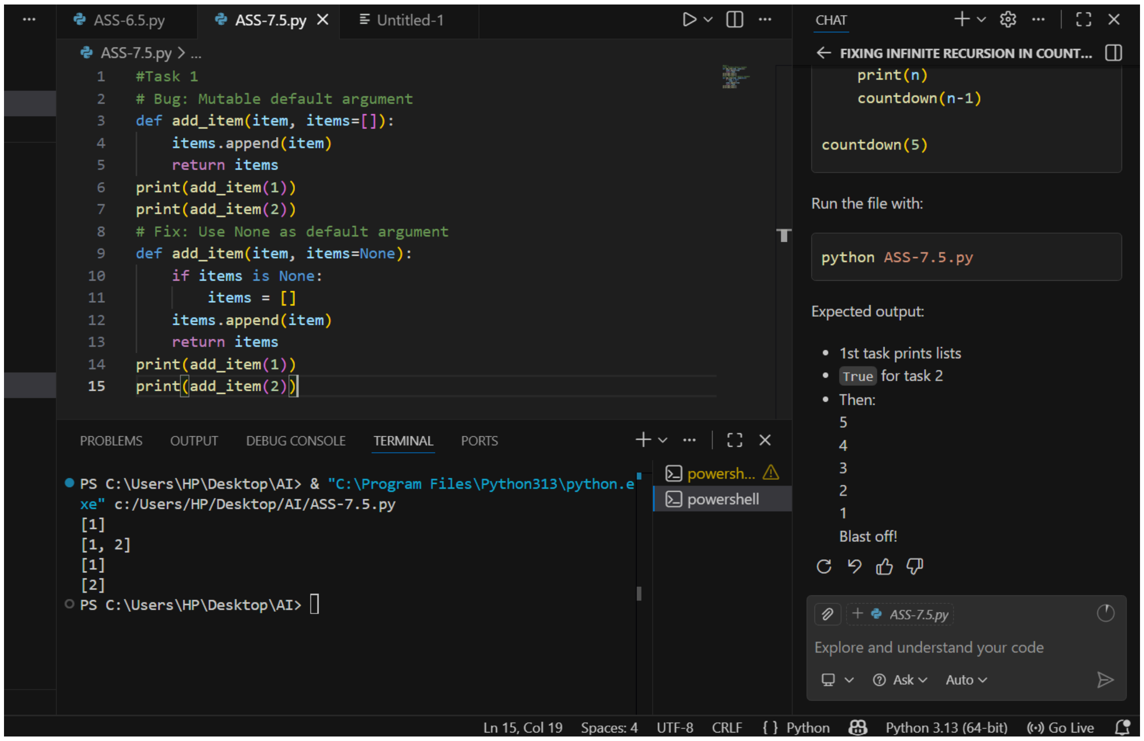
RUDROJU RUPA SRI

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.



```
#Task 1
# Bug: Mutable default argument
def add_item(item, items=[]):
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
# Fix: Use None as default argument
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
print(add_item(1))
print(add_item(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]
PS C:\Users\HP\Desktop\AI>
```

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!

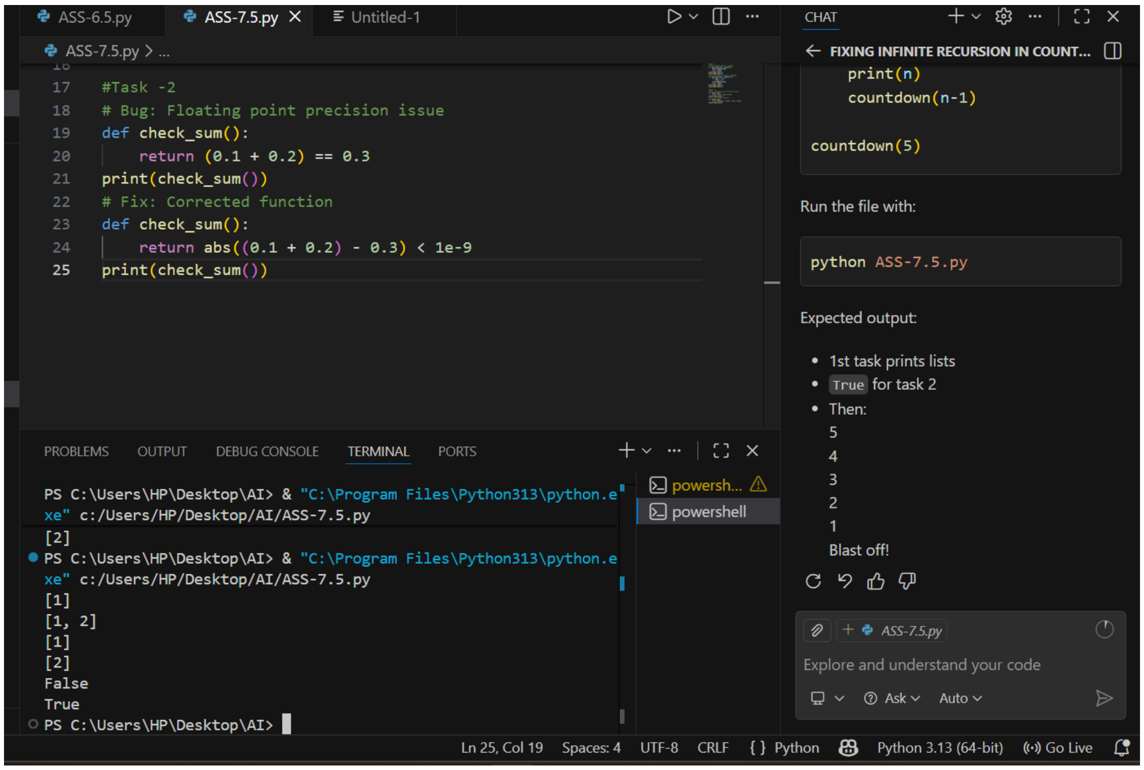
Explore and understand your code

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



```
#Task -2
# Bug: Floating point precision issue
def check_sum():
    return (0.1 + 0.2) == 0.3
print(check_sum())
# Fix: Corrected function
def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-9
print(check_sum())
```

```
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>
```

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!

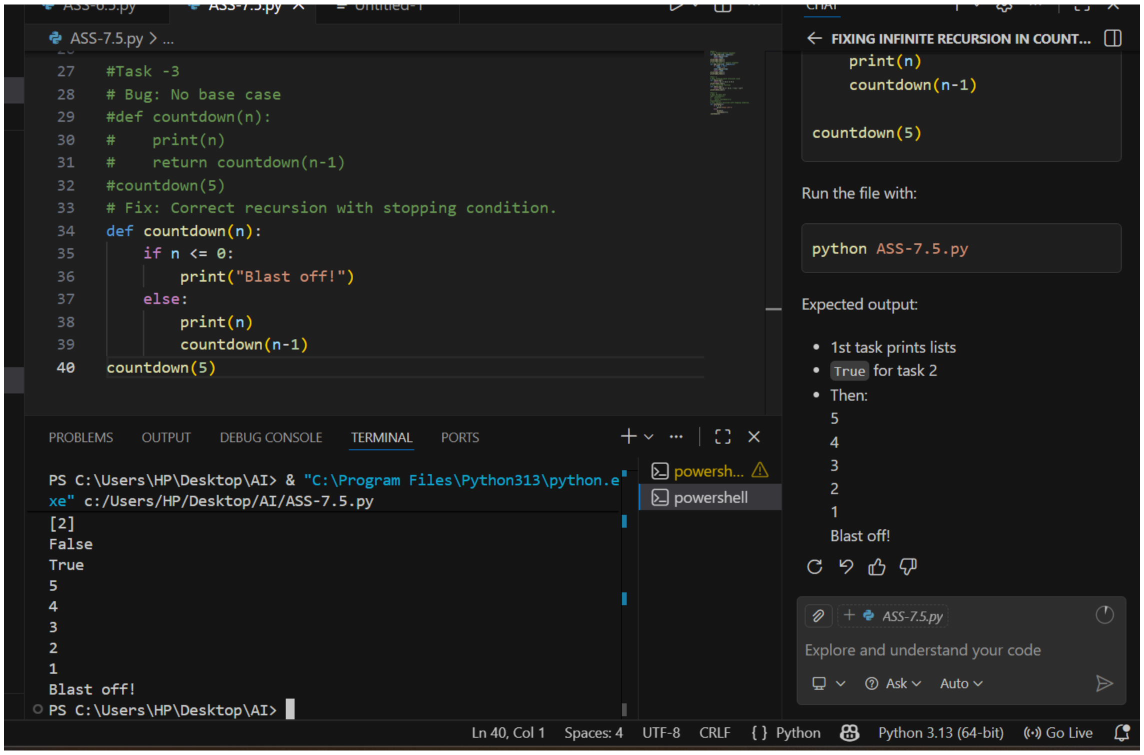
Task 3 (Recursion Error – Missing Base Case)

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.



```
#Task -3
# Bug: No base case
#def countdown(n):
#    print(n)
#    return countdown(n-1)
#countdown(5)
# Fix: Correct recursion with stopping condition.
def countdown(n):
    if n <= 0:
        print("Blast off!")
    else:
        print(n)
        countdown(n-1)
countdown(5)
```

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

FIXING INFINITE RECURSION IN COUNT...

```
    print(n)
    countdown(n-1)

countdown(5)
```

Run the file with:

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

Expected output:

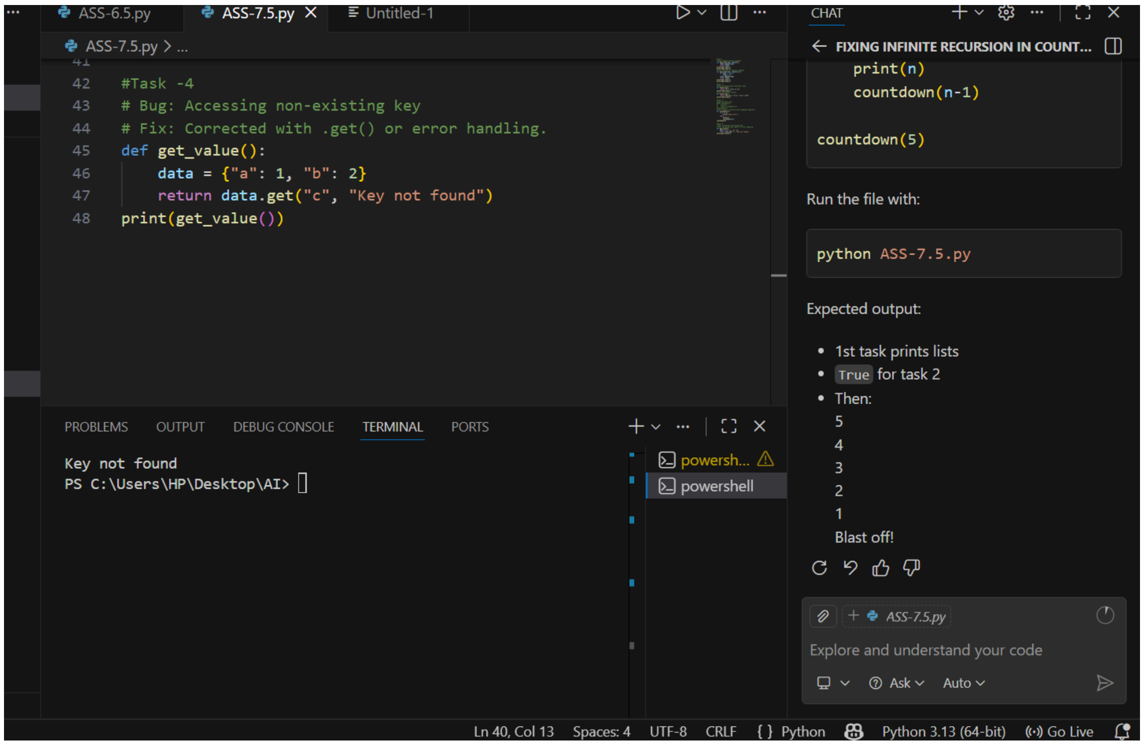
- 1st task prints lists
- `True` for task 2
- Then:
  5
  4
  3
  2
  1
  Blast off!

ASS-7.5.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powersh...
powershell
ASS-7.5.py
Explore and understand your code
Ask
Auto
Ln 40, Col 1 Spaces: 4 UTF-8 CRLF { } Python Python 3.13 (64-bit) Go Live

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.



```python
#Task -4
# Bug: Accessing non-existing key
# Fix: Corrected with .get() or error handling.
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")
print(get_value())
```

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

```
        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 5 (Infinite Loop – Wrong Condition)
Task: Analyze given code where loop never ends. Use AI to detect and fix it.

Bug: Infinite loop

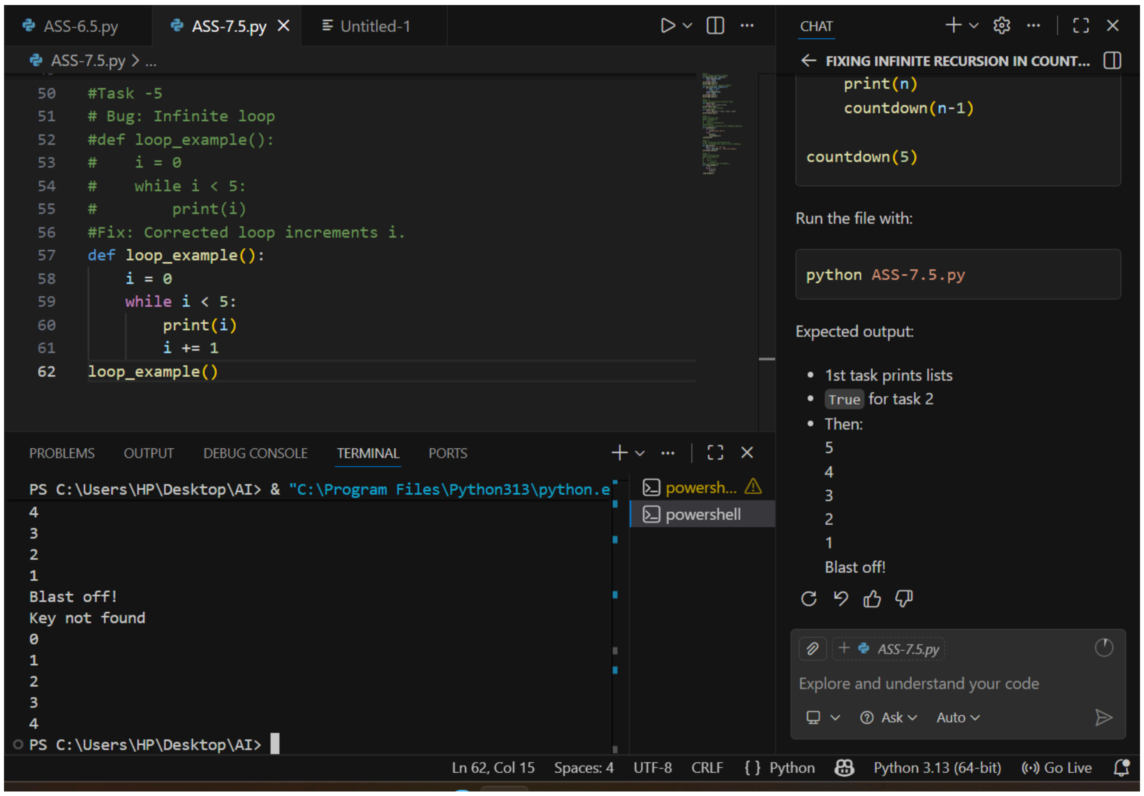
def loop_example():

i = 0

while i < 5:

print(i)

Expected Output: Corrected loop increments i.



```
#Task -5
# Bug: Infinite loop
#def loop_example():
#    i = 0
#    while i < 5:
#        print(i)
#Fix: Corrected loop increments i.
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1
loop_example()
```

```
PS C:\Users\HP\Desktop\AI> & "C:\Program Files\Python313\python.e
4
3
2
1
Blast off!
Key not found
0
1
2
3
4
PS C:\Users\HP\Desktop\AI>
```

```
    print(n)
    countdown(n-1)

countdown(5)
```

Run the file with:

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

Expected output:

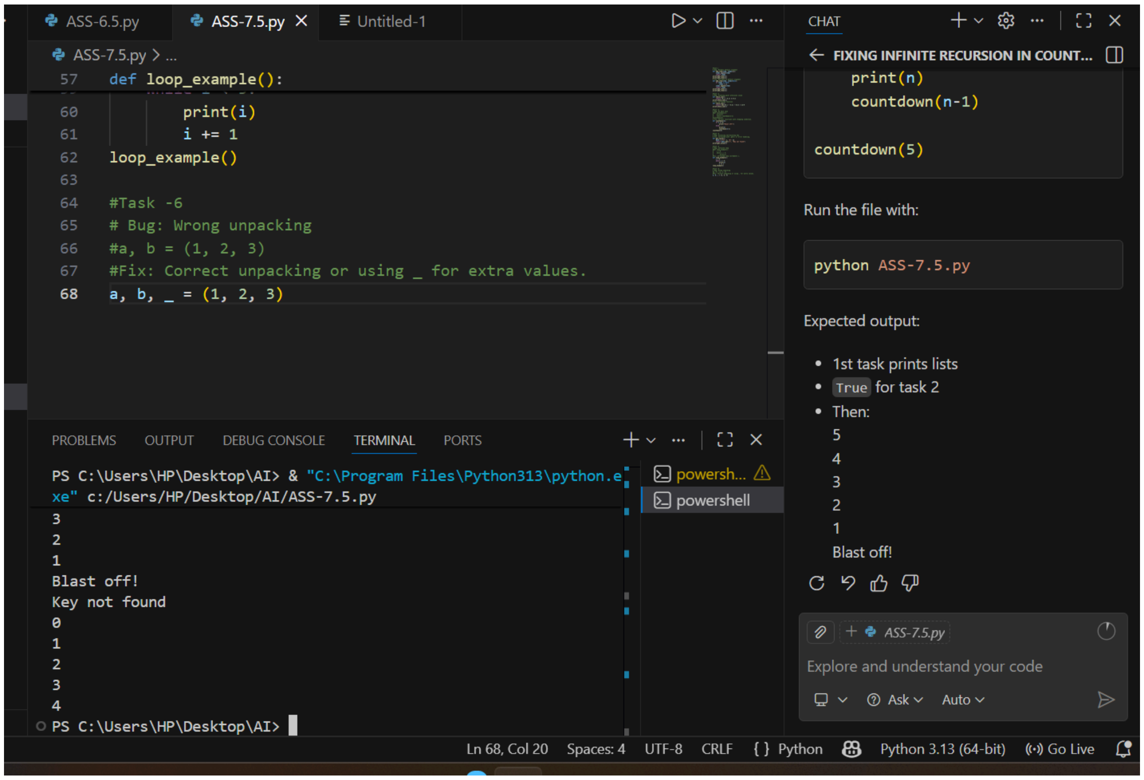
- 1st task prints lists
- `True` for task 2
- Then:
  5
  4
  3
  2
  1
  Blast off!

Task 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.



```python
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)
```

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

FIXING INFINITE RECURSION IN COUNT...

```python
        print(n)
        countdown(n-1)

countdown(5)
```

Run the file with:

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

Expected output:

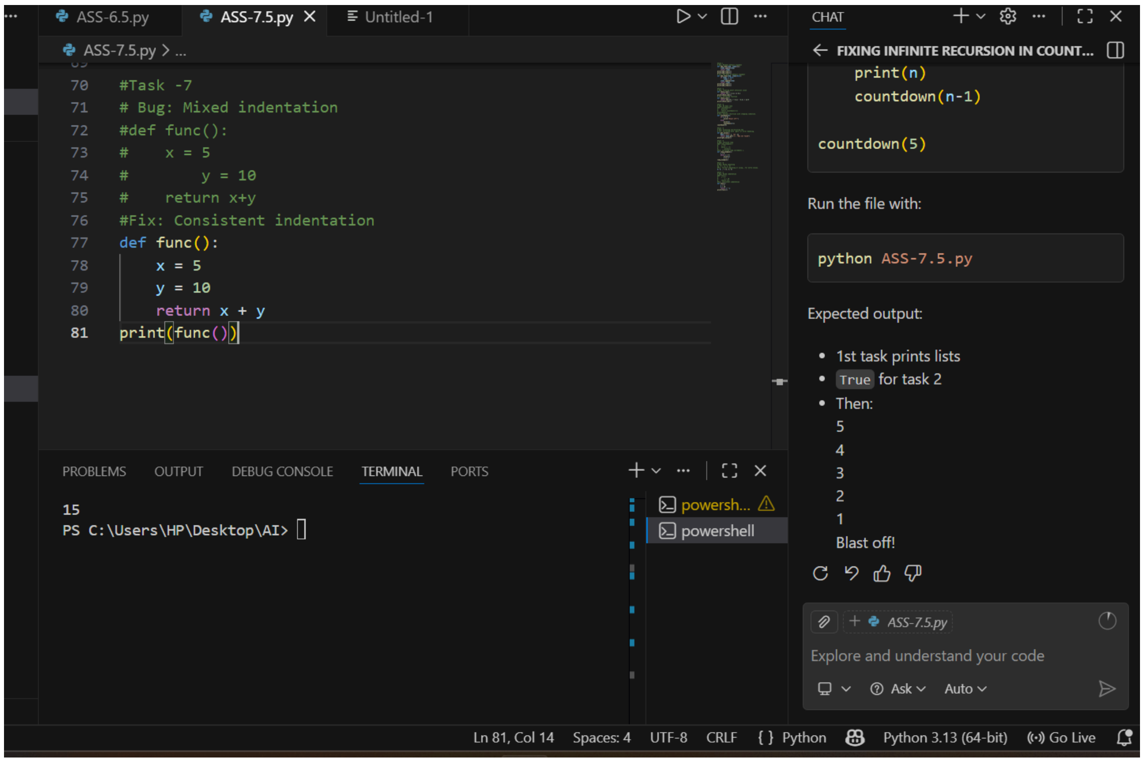
- 1st task prints lists
- `True` for task 2
- Then:
  5
  4
  3
  2
  1
  Blast off!

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.



```
#Task -7
# Bug: Mixed indentation
#def func():
#    x = 5
#        y = 10
#    return x+y
#Fix: Consistent indentation
def func():
    x = 5
    y = 10
    return x + y
print(func())
```

```
15
PS C:\Users\HP\Desktop\AI>
```

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!

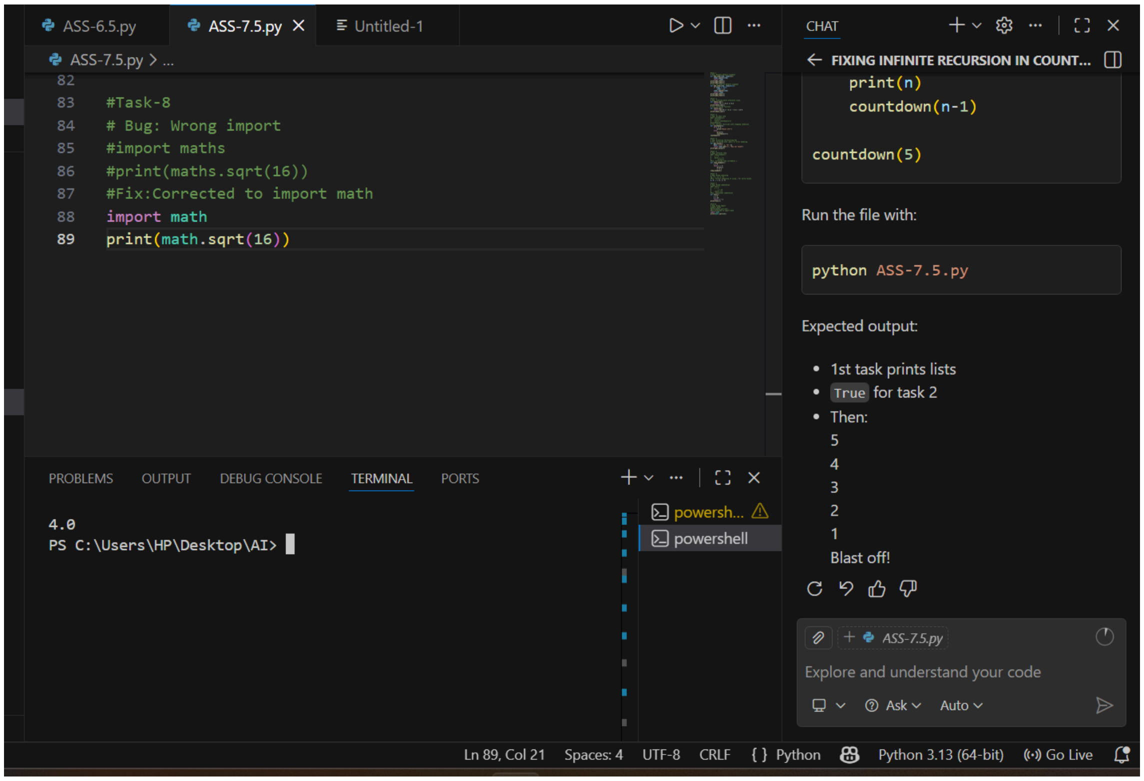
Explore and understand your code

Ln 81, Col 14 Spaces: 4 UTF-8 CRLF { } Python Python 3.13 (64-bit) Go Live

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



```
#Task-8
# Bug: Wrong import
#import maths
#print(maths.sqrt(16))
#Fix:Corrected to import math
import math
print(math.sqrt(16))
```

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

FIXING INFINITE RECURSION IN COUNT...

```
    print(n)
    countdown(n-1)

countdown(5)
```

Run the file with:

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

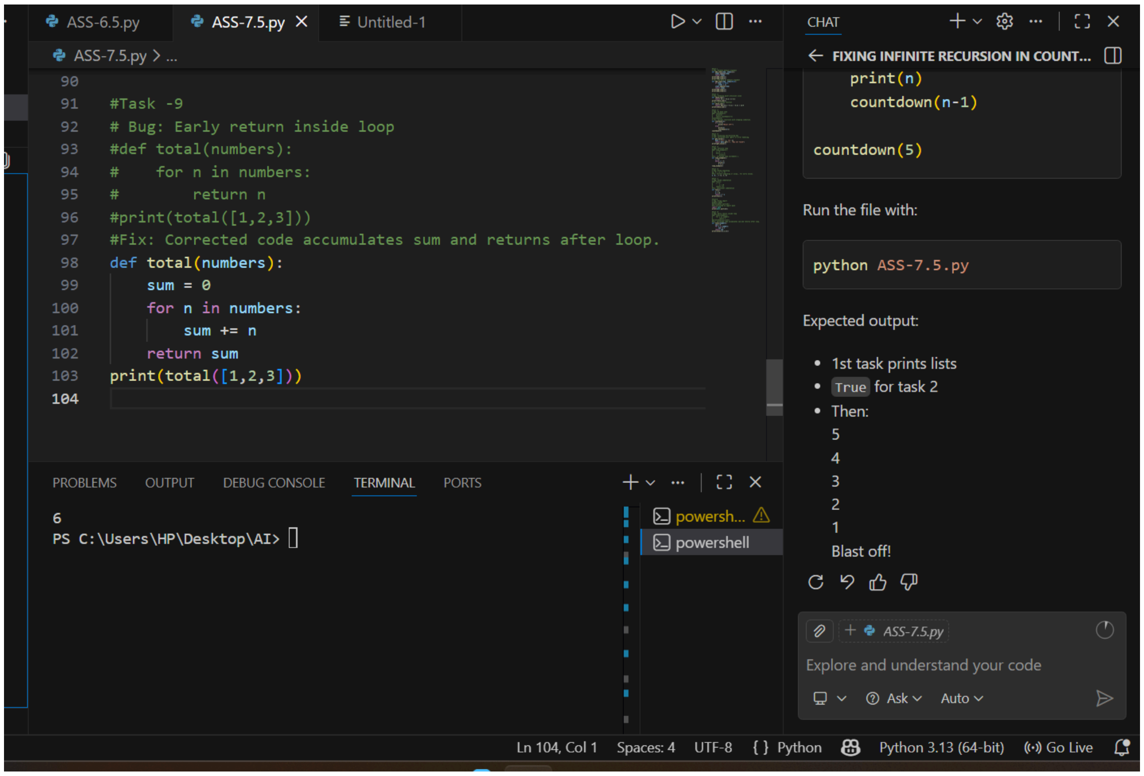
Expected output:

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

Task 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.



```python
#Task -9
# Bug: Early return inside loop
#def total(numbers):
#    for n in numbers:
#        return n
#print(total([1,2,3]))
#Fix: Corrected code accumulates sum and returns after loop.
def total(numbers):
    sum = 0
    for n in numbers:
        sum += n
    return sum
print(total([1,2,3]))
```

```
6
PS C:\Users\HP\Desktop\AI>
```

```python
    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!

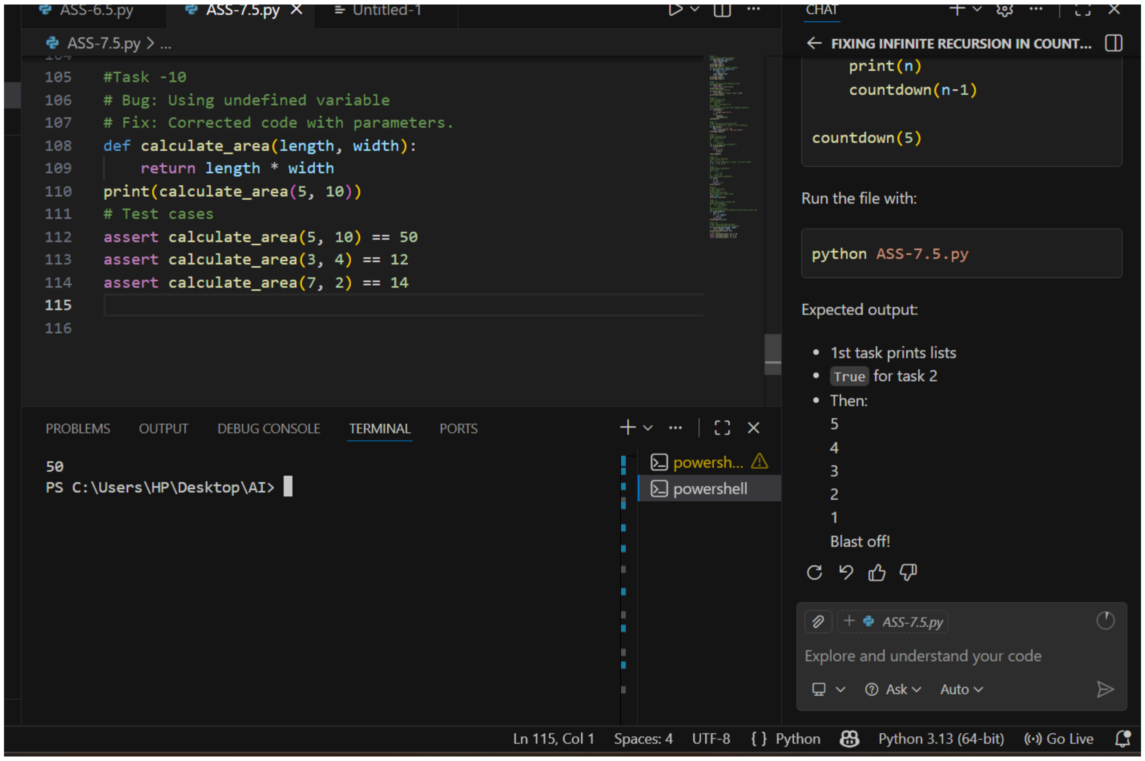
Task 10 (Name Error – Undefined Variable)
Task: Analyze given code where a variable is used before being defined. Let AI detect and fix the error.

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.



```
#Task -10
# Bug: Using undefined variable
# Fix: Corrected code with parameters.
def calculate_area(length, width):
    return length * width
print(calculate_area(5, 10))
# Test cases
assert calculate_area(5, 10) == 50
assert calculate_area(3, 4) == 12
assert calculate_area(7, 2) == 14
```

```
50
PS C:\Users\HP\Desktop\AI>
```

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!

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.

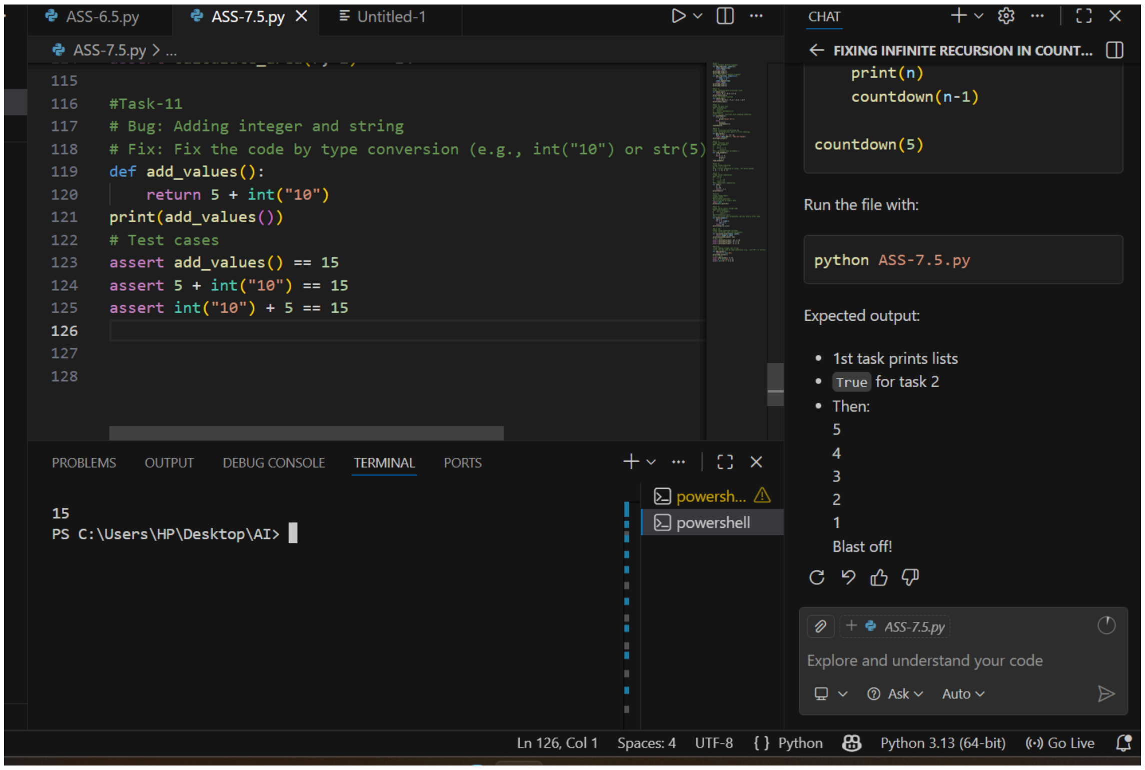
Task 11 (Type Error – Mixing Data Types Incorrectly)
Task: Analyze given code where integers and strings are added incorrectly. Let AI detect and fix the error.

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.



```
#Task-11
# Bug: Adding integer and string
# Fix: Fix the code by type conversion (e.g., int("10") or str(5)
def add_values():
    return 5 + int("10")
print(add_values())
# Test cases
assert add_values() == 15
assert 5 + int("10") == 15
assert int("10") + 5 == 15
```

```
15
PS C:\Users\HP\Desktop\AI>
```

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!

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.

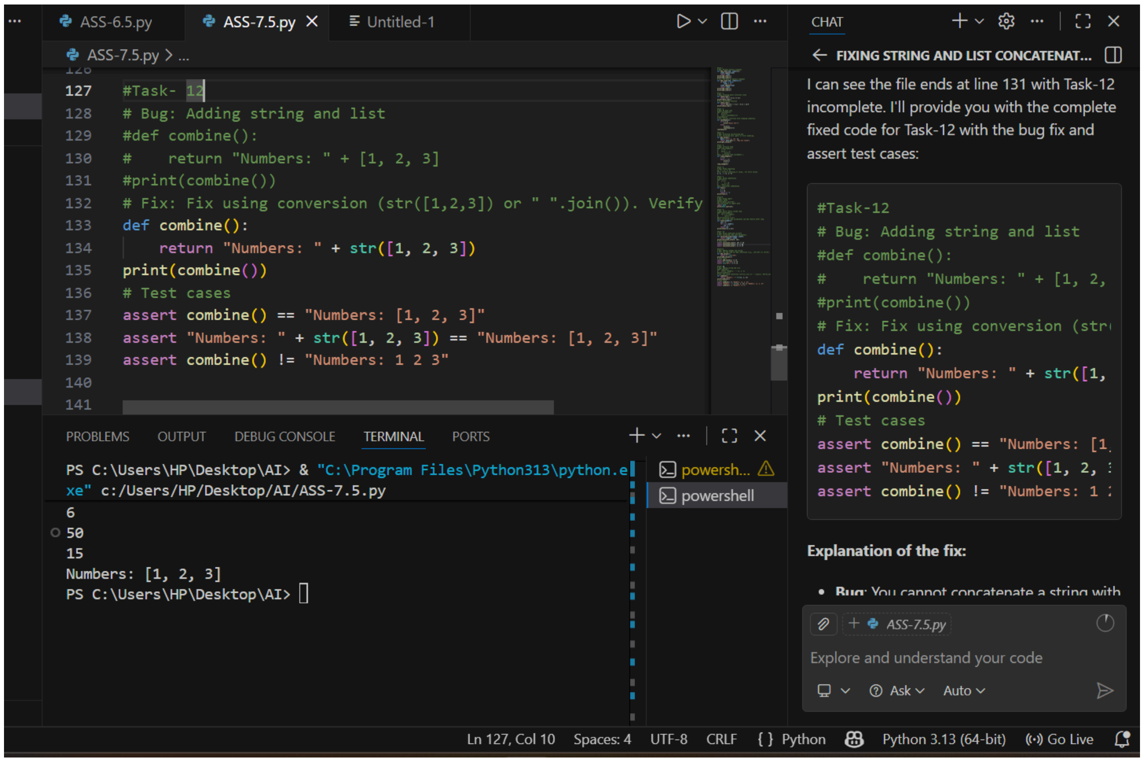
Bug: Adding string and list

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



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

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

FIXING STRING AND LIST CONCATENAT...

I can see the file ends at line 131 with Task-12 incomplete. I'll provide you with the complete fixed code for Task-12 with the bug fix and assert test cases:

```python
#Task-12
# Bug: Adding string and list
#def combine():
#    return "Numbers: " + [1, 2,
#print(combine())
# Fix: Fix using conversion (str
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

Explore and understand your code

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.

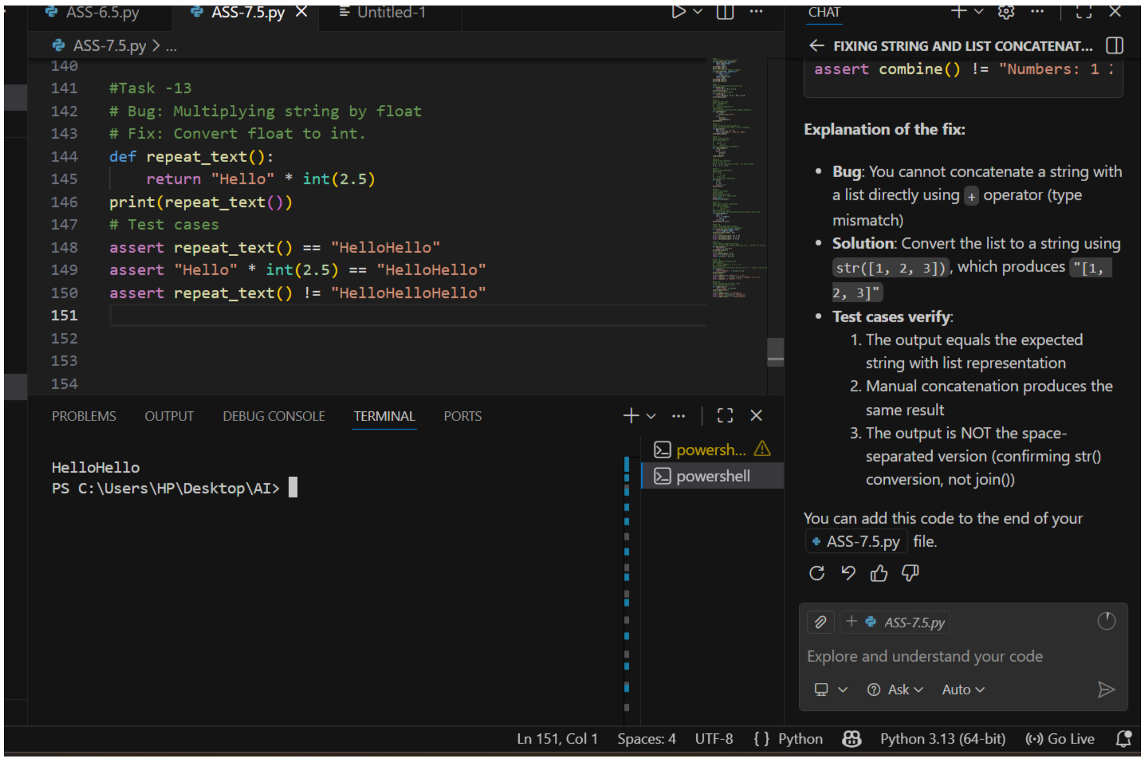
Task 13 (Type Error – Multiplying String by Float)
Task: Detect and fix code where a string is multiplied by a float.

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.



```python
#Task -13
# Bug: Multiplying string by float
# Fix: Convert float to int.
def repeat_text():
    return "Hello" * int(2.5)
print(repeat_text())
# Test cases
assert repeat_text() == "HelloHello"
assert "Hello" * int(2.5) == "HelloHello"
assert repeat_text() != "HelloHelloHello"
```

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

```
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.

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.

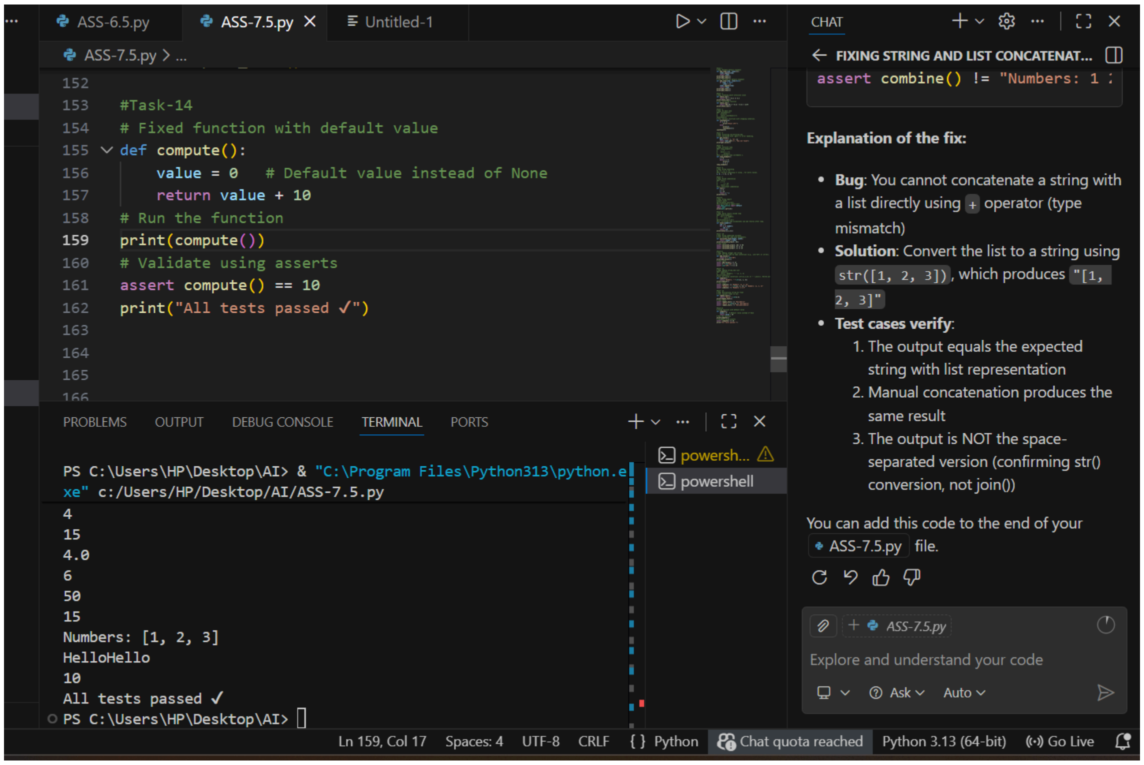
Task 14 (Type Error – Adding None to Integer)
Task: Analyze code where None is added to an integer.

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.



```python
#Task-14
# Fixed function with default value
def compute():
    value = 0   # Default value instead of None
    return value + 10
# Run the function
print(compute())
# Validate using asserts
assert compute() == 10
print("All tests passed ✓")
```

```
PS C:\Users\HP\Desktop\AI> & "C:\Program Files\Python313\python.e
xe" 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> 
```

```
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.

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).

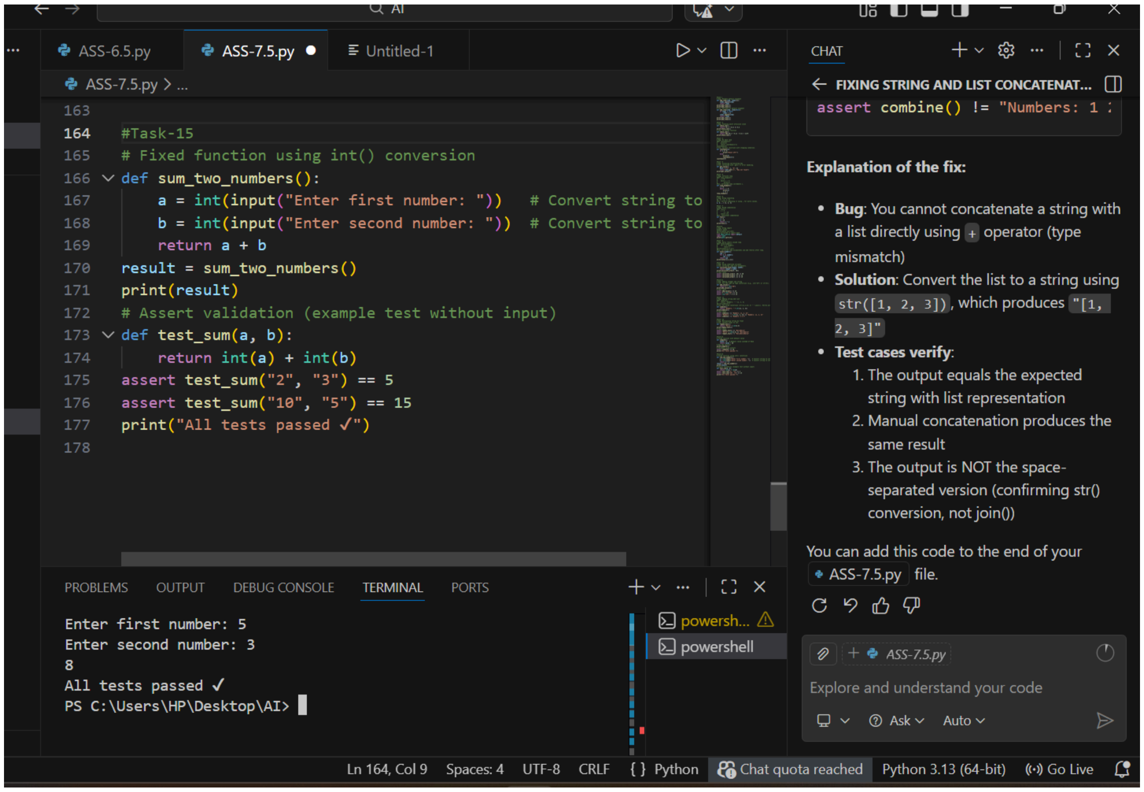
Task 15 (Type Error – Input Treated as String Instead of Number)
Task: Fix code where user input is not converted properly.

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.



```python
#Task-15
# Fixed function using int() conversion
def sum_two_numbers():
    a = int(input("Enter first number: "))   # Convert string to
    b = int(input("Enter second number: "))  # Convert string to
    return a + b
result = sum_two_numbers()
print(result)
# Assert validation (example test without input)
def test_sum(a, b):
    return int(a) + int(b)
assert test_sum("2", "3") == 5
assert test_sum("10", "5") == 15
print("All tests passed ✓")
```

```
Enter first number: 5
Enter second number: 3
8
All tests passed ✓
PS C:\Users\HP\Desktop\AI>
```

FIXING STRING AND LIST CONCATENAT...

```
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.

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.