ASSIGNMENT 7.5

Name: A.Sai Varshith

Roll Number : 2303A51831

Batch – 04

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)

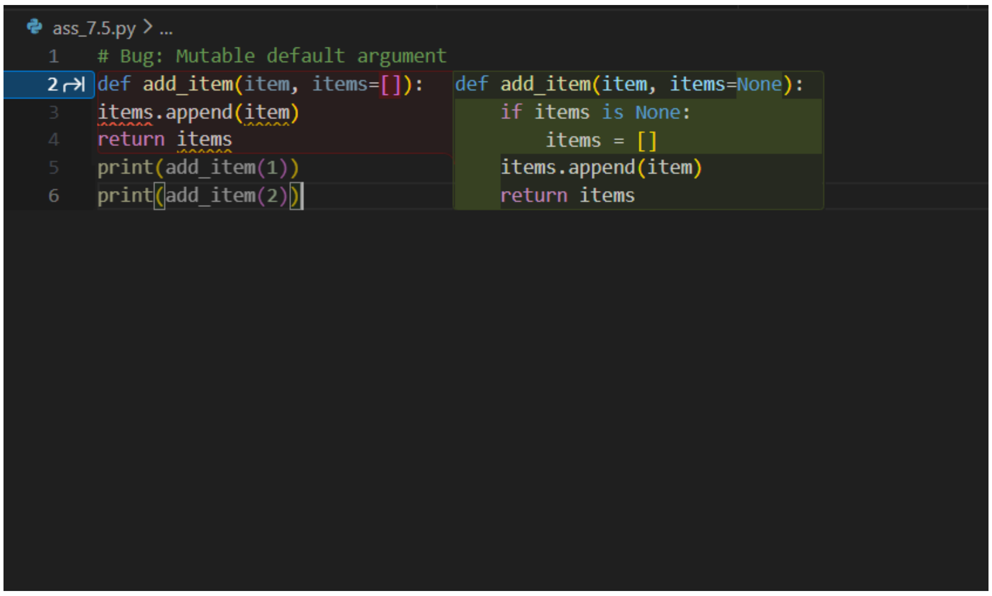
return items

print(add_item(1))

print(add_item(2))

Expected Output: Corrected function avoids shared list bug.

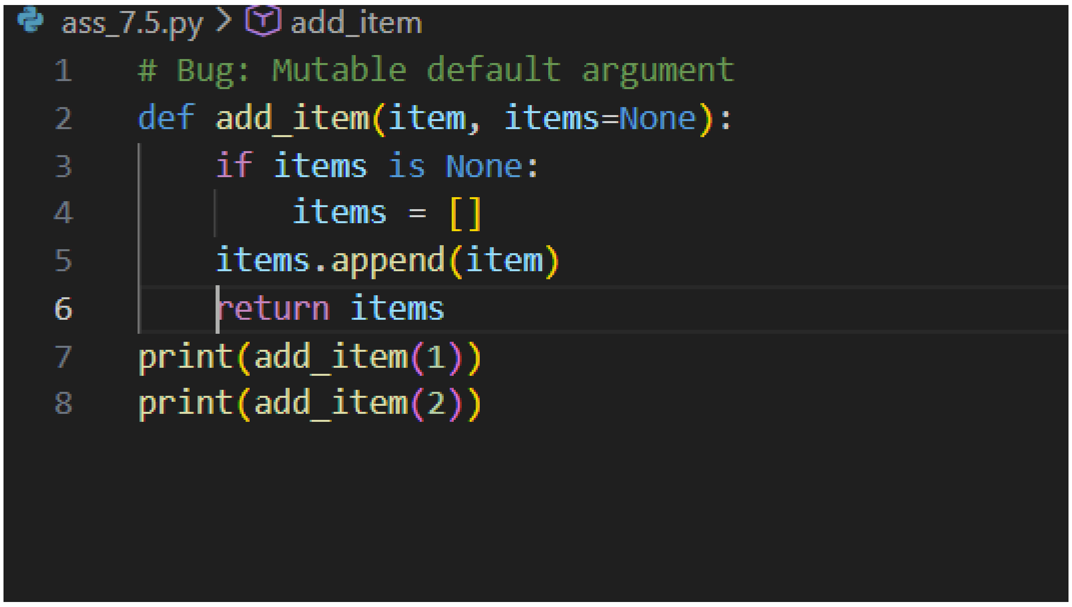
INPUT:



ass_7.5.py > ...

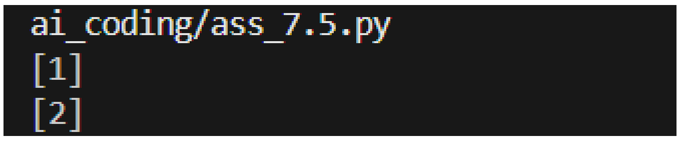
```
# Bug: Mutable default argument
def add_item(item, items=[]):
items.append(item)
return items
print(add_item(1))
print(add_item(2))
```

```
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
```



```
ass_7.5.py > add_item
# Bug: Mutable 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))
```

OUTPUT:



```
ai_coding/ass_7.5.py
[1]
[2]
```

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

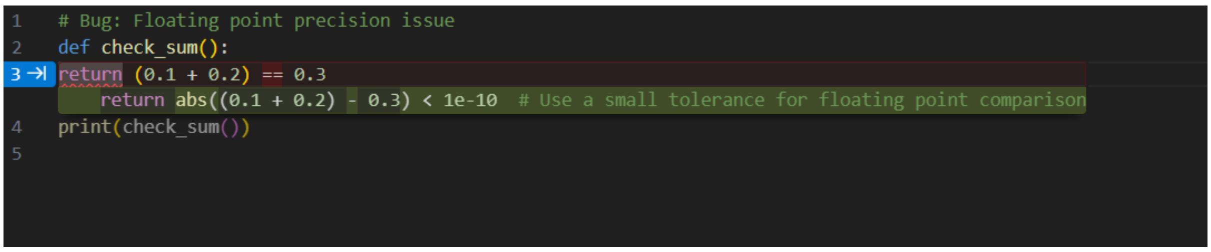
def check_sum():

return (0.1 + 0.2) == 0.3

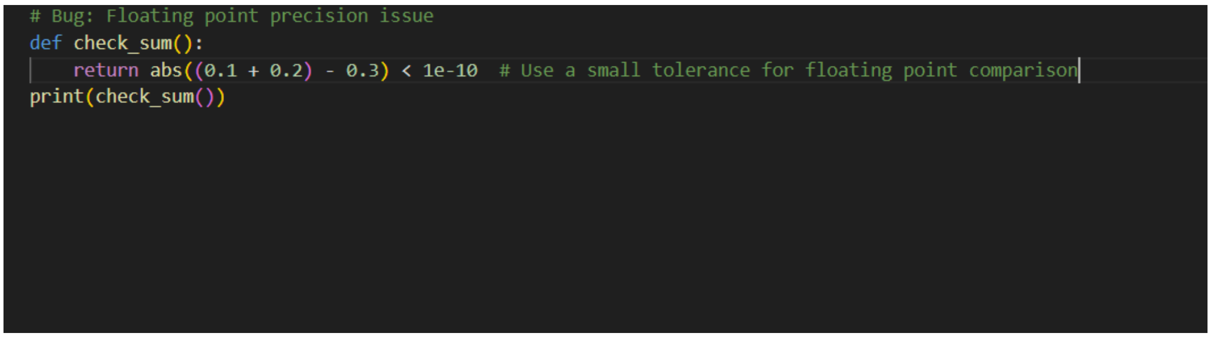
print(check_sum())

Expected Output: Corrected function

INPUT:

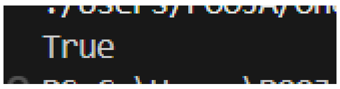


```
# Bug: Floating point precision issue
def check_sum():
return (0.1 + 0.2) == 0.3
    return abs((0.1 + 0.2) - 0.3) < 1e-10  # Use a small tolerance for floating point comparison
print(check_sum())
```



```python
# Bug: Floating point precision issue
def check_sum():
    return abs((0.1 + 0.2) - 0.3) < 1e-10  # Use a small tolerance for floating point comparison
print(check_sum())
```

OUTPUT:



```
True
```

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):

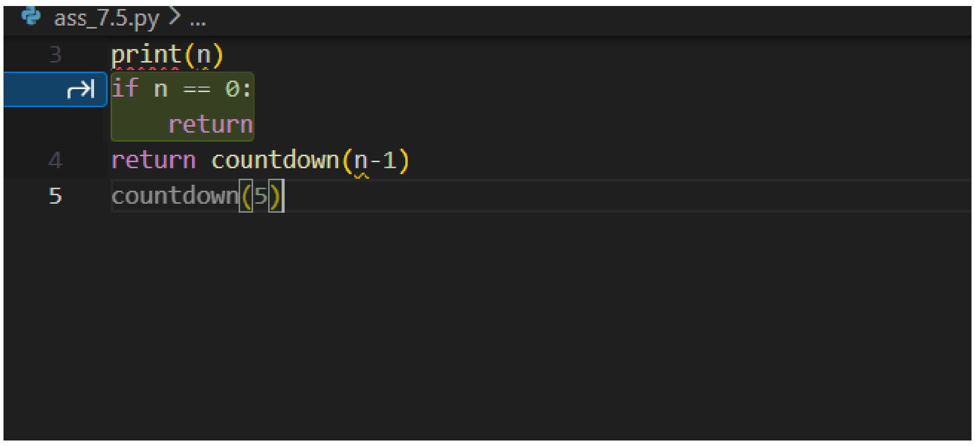
print(n)

return countdown(n-1)

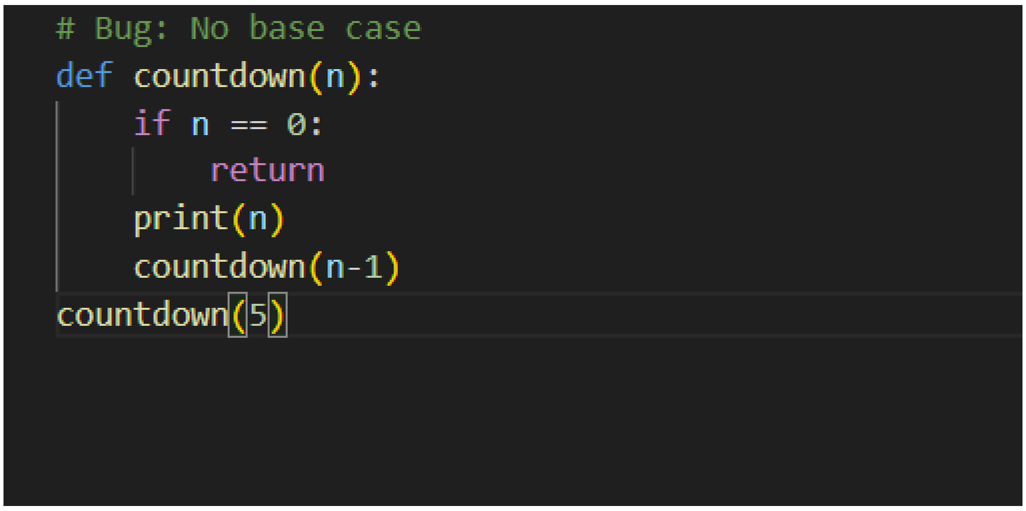
countdown(5)

Expected Output : Correct recursion with stopping condition

INPUT:

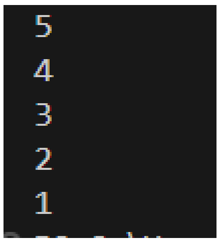


```python
print(n)
if n == 0:
    return
return countdown(n-1)
countdown(5)
```



```python
# Bug: No base case
def countdown(n):
    if n == 0:
        return
    print(n)
    countdown(n-1)
countdown(5)
```

OUTPUT:



```
5
4
3
2
1
```

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():

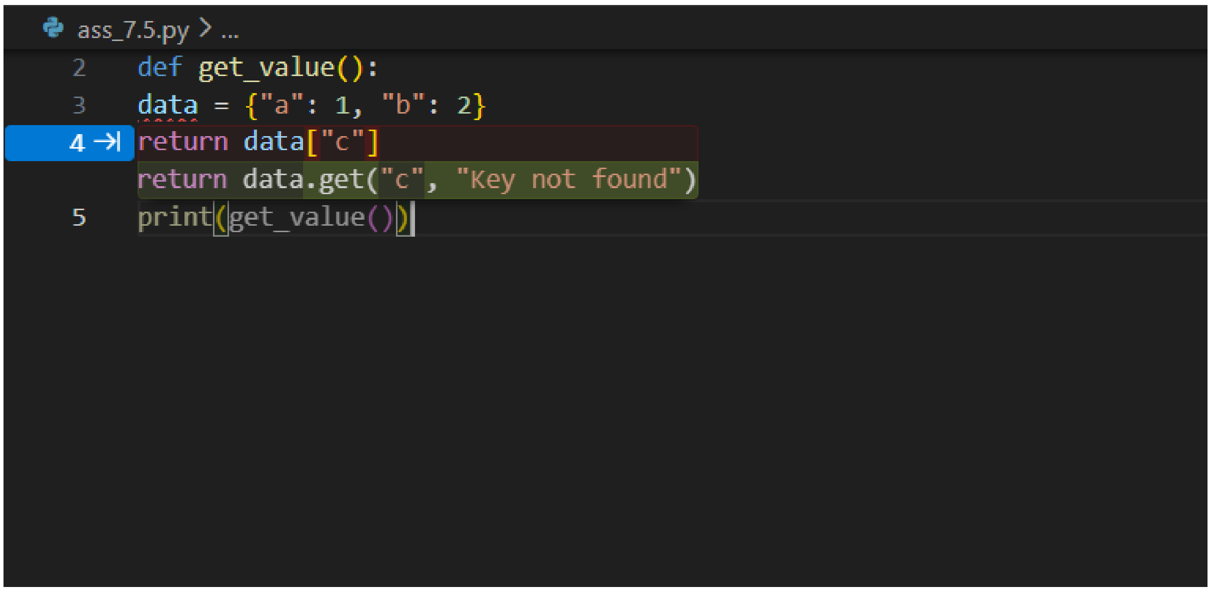
data = {"a": 1, "b": 2}

return data["c"]

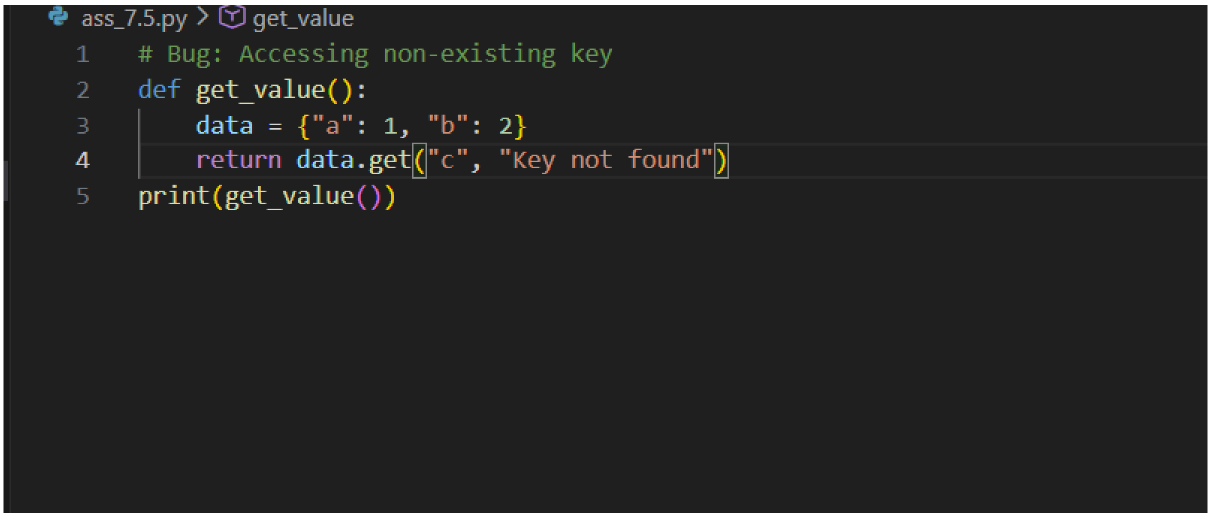
print(get_value())

Expected Output: Corrected with .get() or error handling.

INPUT:

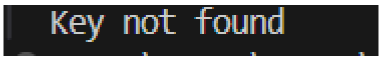


```
def get_value():
data = {"a": 1, "b": 2}
return data["c"]
return data.get("c", "Key not found")
print(get_value())
```



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

OUTPUT:



```
Key not found
```

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():

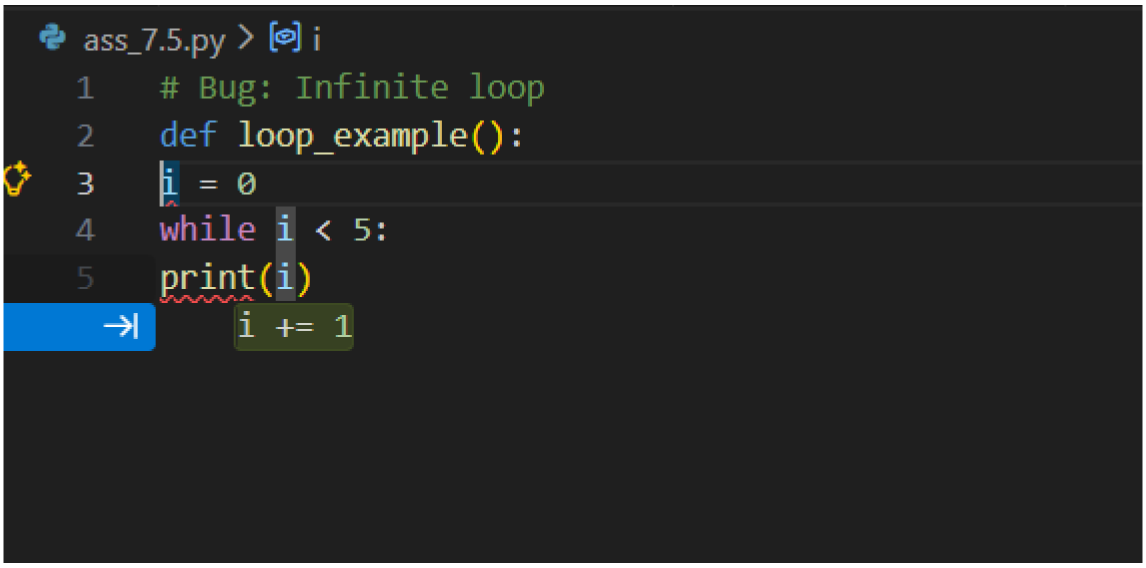
i = 0

while i < 5:

print(i)

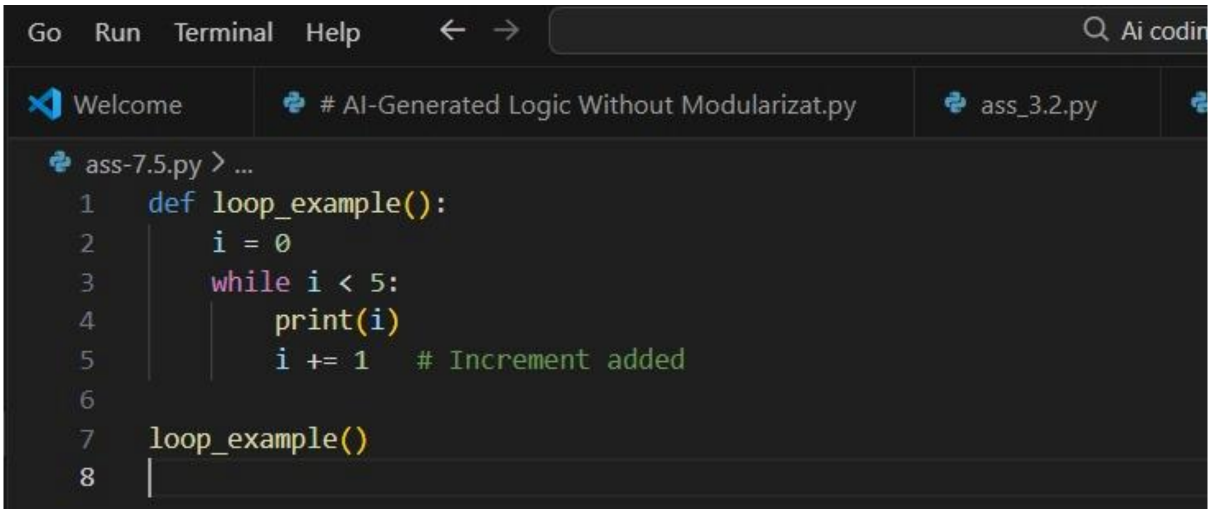
Expected Output: Corrected loop increments i.

INPUT:



ass_7.5.py > i

```
# Bug: Infinite loop
def loop_example():
i = 0
while i < 5:
print(i)
    i += 1
```

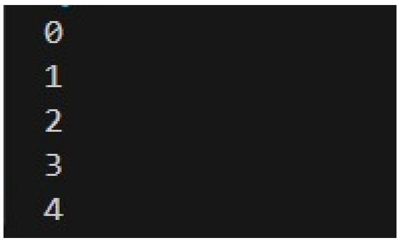


ass-7.5.py > ...

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1   # Increment added

loop_example()
```

OUTPUT:



```
0
1
2
3
4
```

Task 6 (Unpacking Error – Wrong Variables)

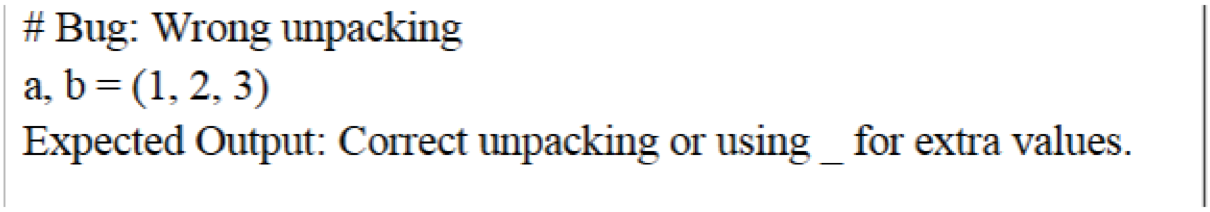
Task: Analyze given code where tuple unpacking fails. Use AI to fix it.

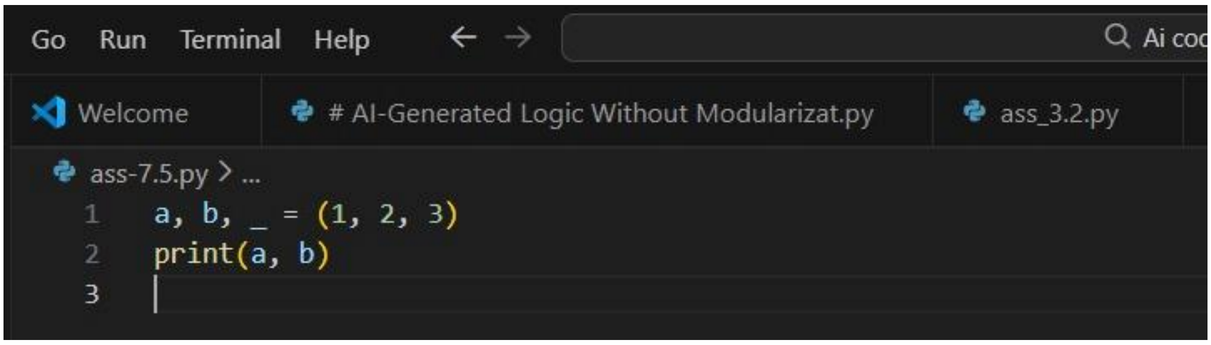
0# Bug: Wrong unpacking

a, b = (1, 2, 3)

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

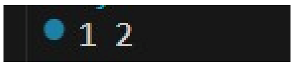
INPUT:


# Bug: Wrong unpacking
a, b = (1, 2, 3)
Expected Output: Correct unpacking or using _ for extra values.



```python
a, b, _ = (1, 2, 3)
print(a, b)
```

OUTPUT:



```
1 2
```

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():

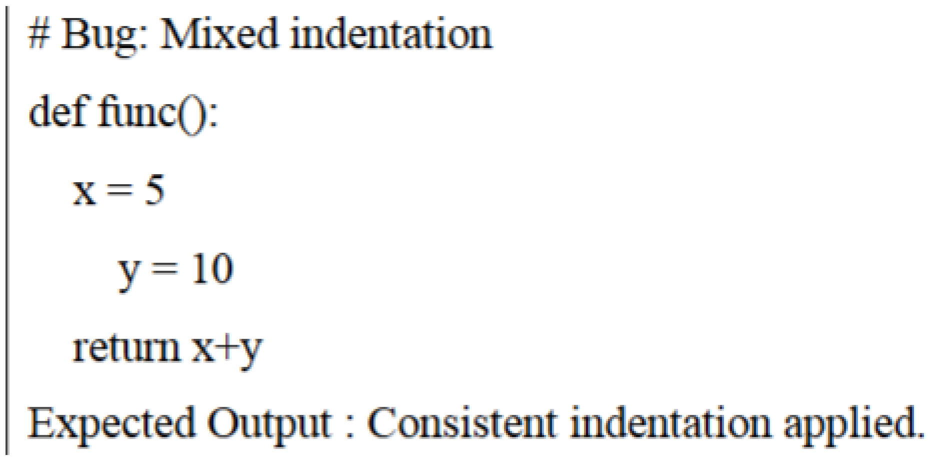
x = 5

y = 10

return x+y

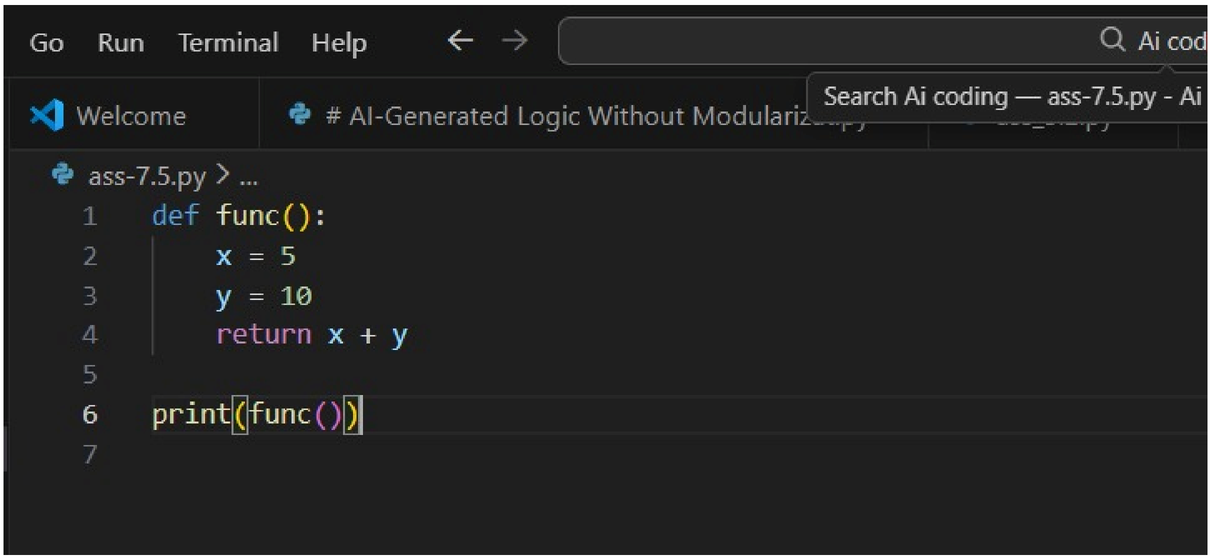
Expected Output : Consistent indentation applied.

INPUT:



```
# Bug: Mixed indentation
def func():
    x = 5
        y = 10
    return x+y
```

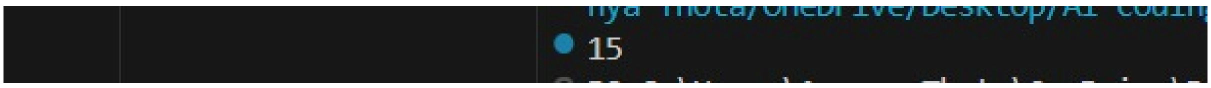
Expected Output : Consistent indentation applied.


Go
Run
Terminal
Help
Ai cod
Search Ai coding — ass-7.5.py - Ai
Welcome
# AI-Generated Logic Without Modulariz
ass-7.5.py > ...

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

print(func())
```

OUTPUT:



```
15
```

Task 8 (Import Error – Wrong Module Usage)

Task: Analyze given code with incorrect import. Use AI to fix.

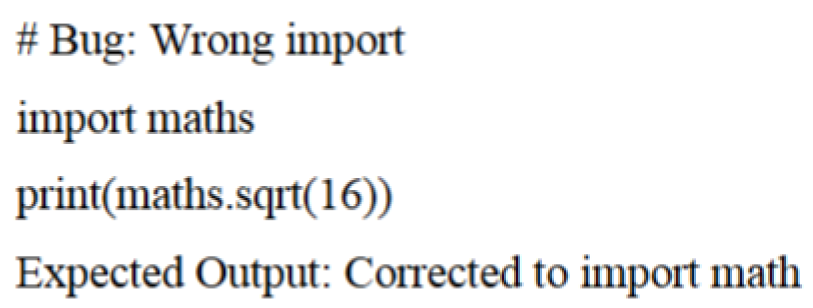
# Bug: Wrong import

import maths

print(maths.sqrt(16))

Expected Output: Corrected to import math

INPUT:

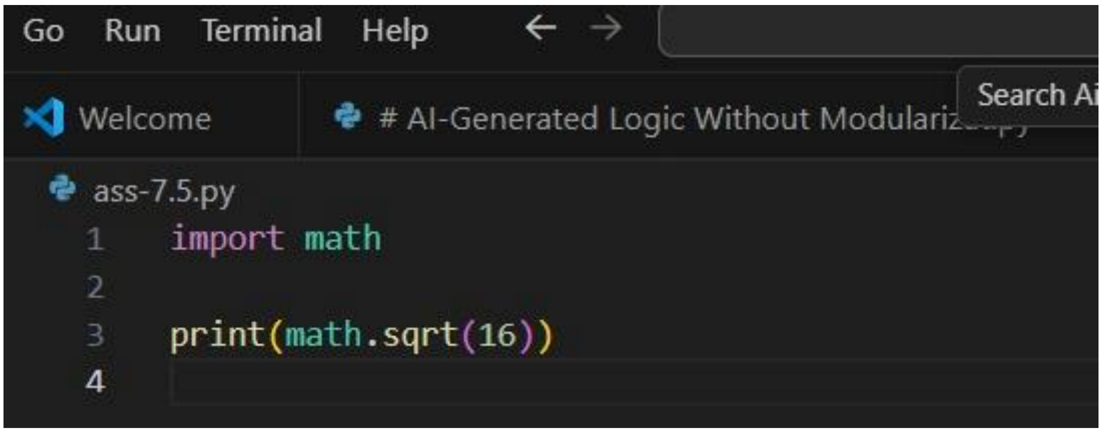


Bug: Wrong import

import maths

print(maths.sqrt(16))

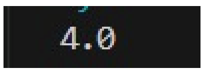
Expected Output: Corrected to import math



```python
import math

print(math.sqrt(16))
```

OUTPUT:



```
4.0
```