Lab Assignment-7.5

Name:P.Chandra Vardhan Reddy

Hallticket:2303A51034

Batch:01

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=[]):

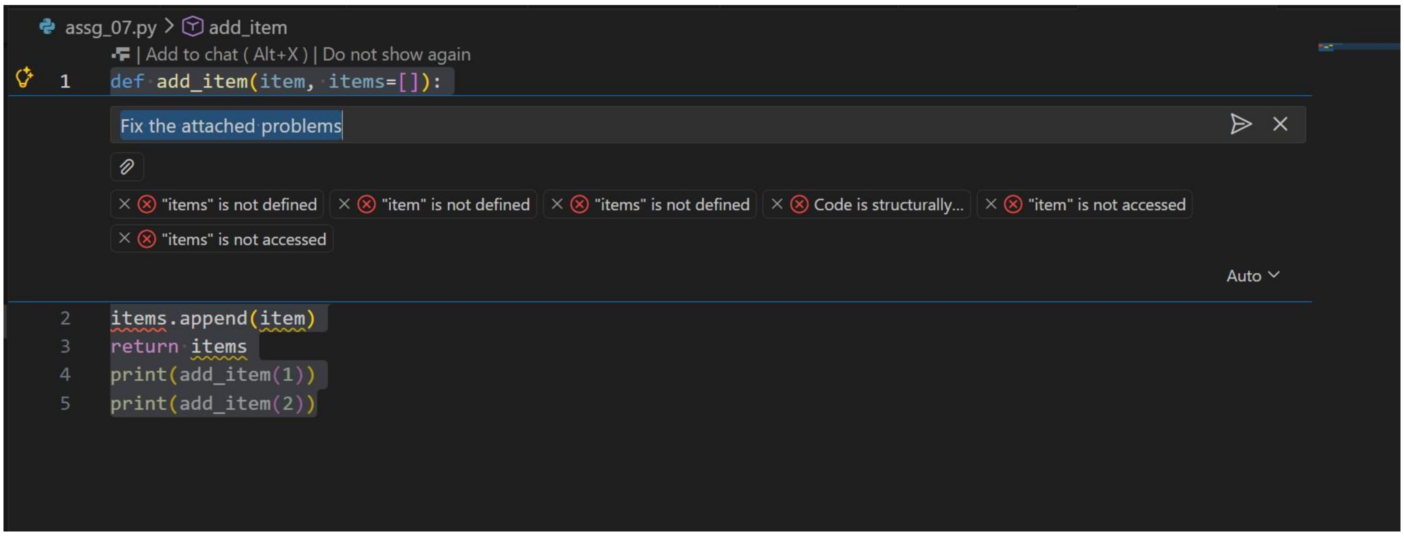
items.append(item) return

items print(add_item(1))

print(add_item(2))

Expected Output: Corrected function avoids shared list bug.

Screenshots:

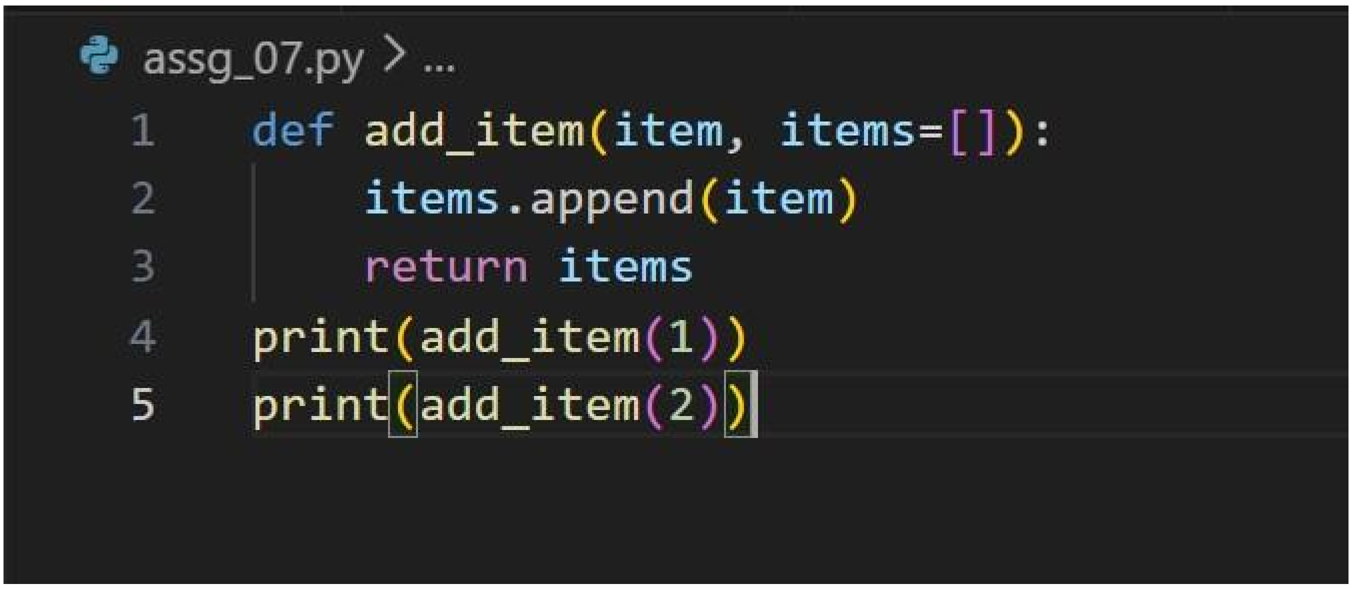


```
def add_item(item, items=[]):
```

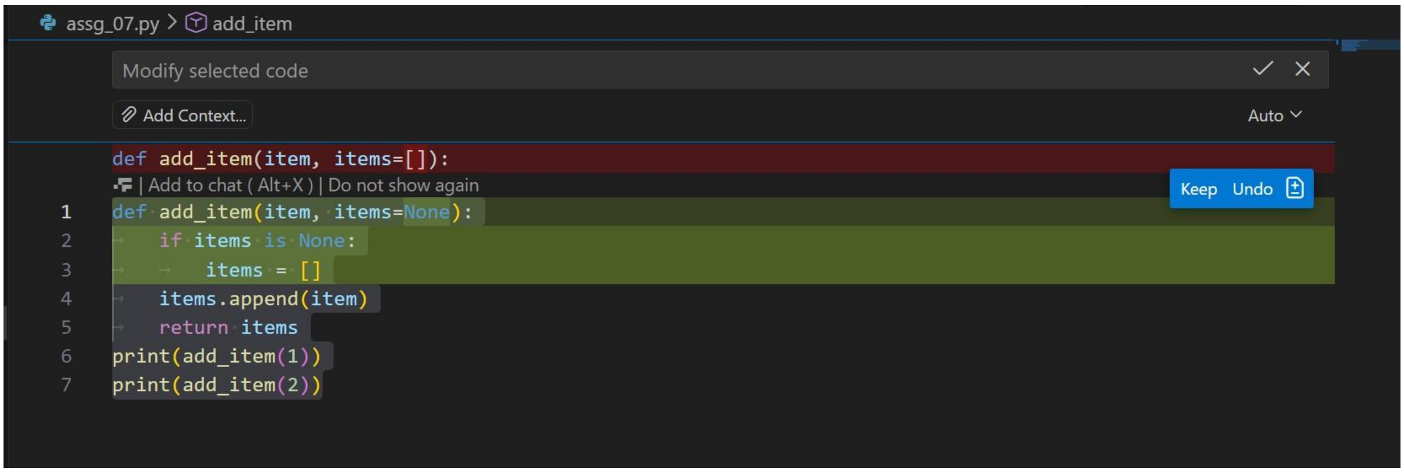
Fix the attached problems

"items" is not defined · "item" is not defined · "items" is not defined · Code is structurally... · "item" is not accessed · "items" is not accessed

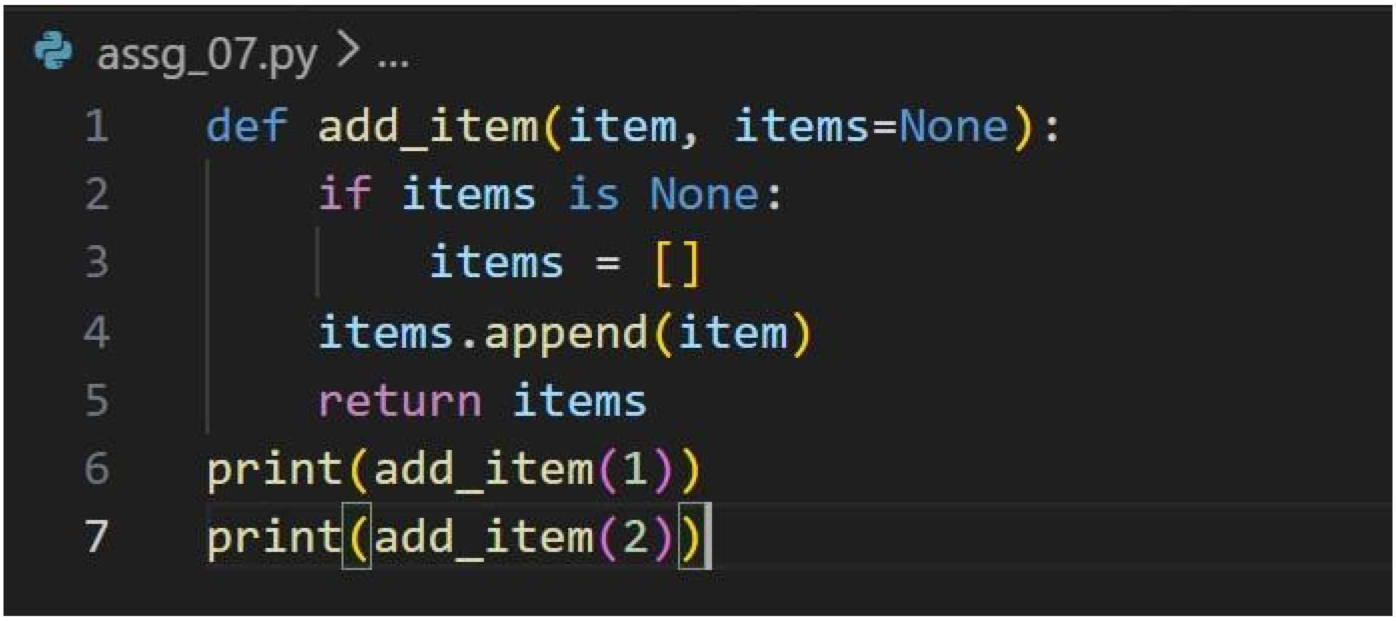
```
items.append(item)
return items
print(add_item(1))
print(add_item(2))
```



```
assg_07.py > ...
def add_item(item, items=[]):
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```



```
def add_item(item, items=[]):
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```

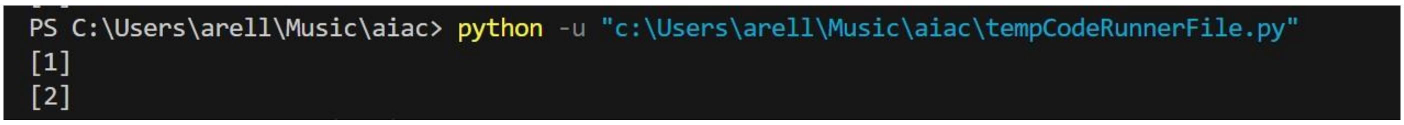


```
assg_07.py > ...
def add_item(item, items=None):
    if items is None:
        items = []
    items.append(item)
    return items
print(add_item(1))
print(add_item(2))
```

<u>Code:</u>

```
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:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\tempCodeRunnerFile.py"
[1]
[2]
```

Task 2 (Floating-Point Precision Error)

Task: Analyze given code where floating-point comparison fails.

Use AI to correct with tolerance. #

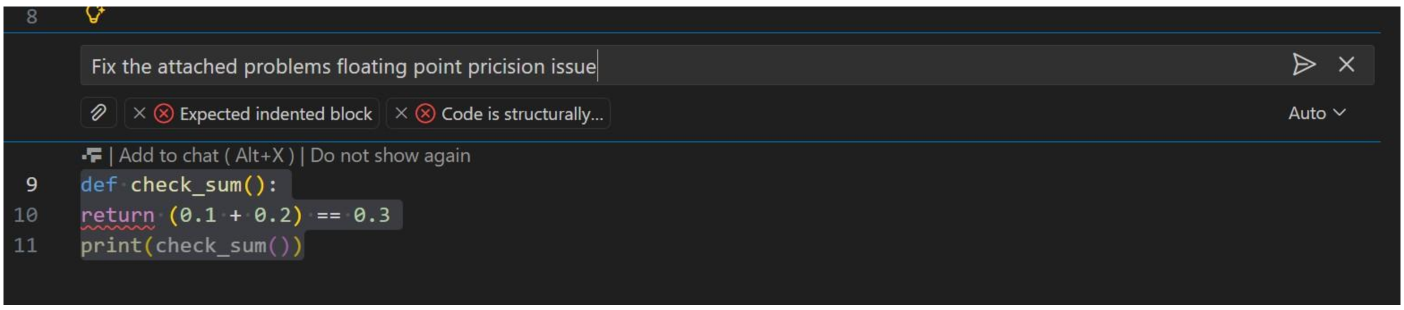
Bug: Floating point precision issue

def check_sum(): return (0.1 + 0.2)

== 0.3 print(check_sum())

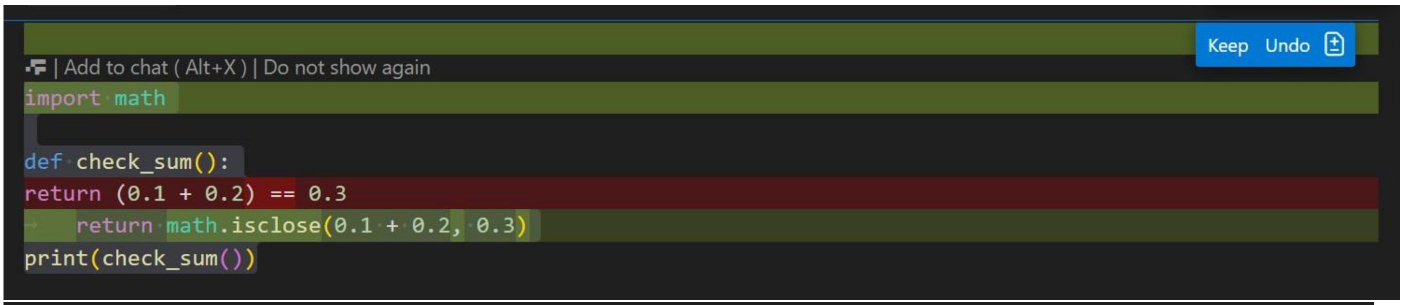
Expected Output: Corrected function

Screenshots:

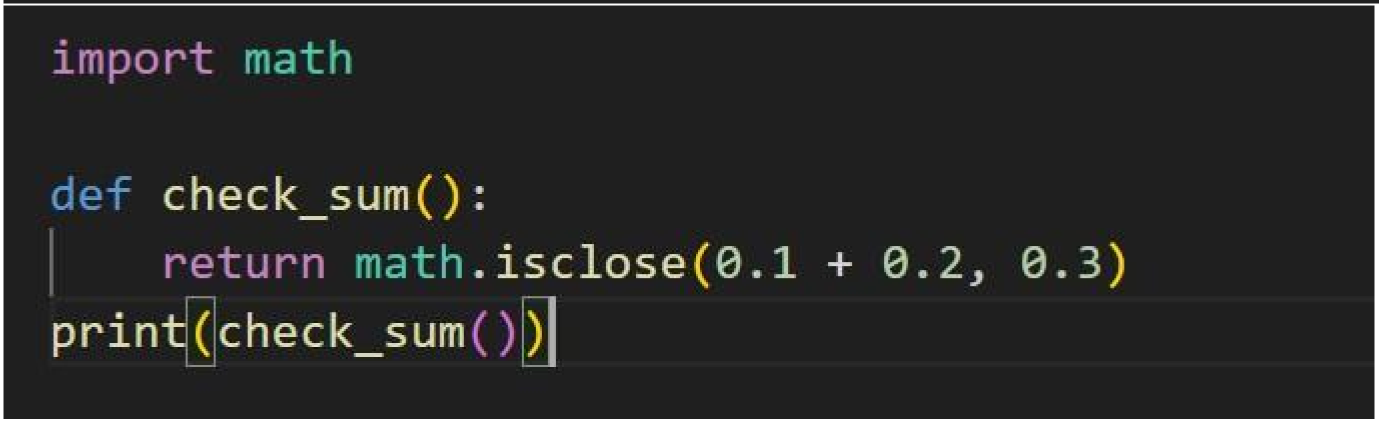


Fix the attached problems floating point pricision issue

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



```
import math

def check_sum():
return (0.1 + 0.2) == 0.3
    return math.isclose(0.1 + 0.2, 0.3)
print(check_sum())
```



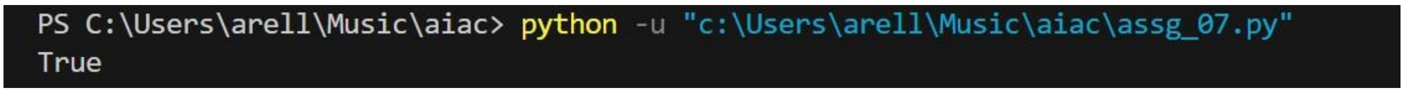
```
import math

def check_sum():
    return math.isclose(0.1 + 0.2, 0.3)
print(check_sum())
```

Code:

```
import math
```

```
def check_sum():     return
math.isclose(0.1 + 0.2, 0.3)
print(check_sum())
```

output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
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):

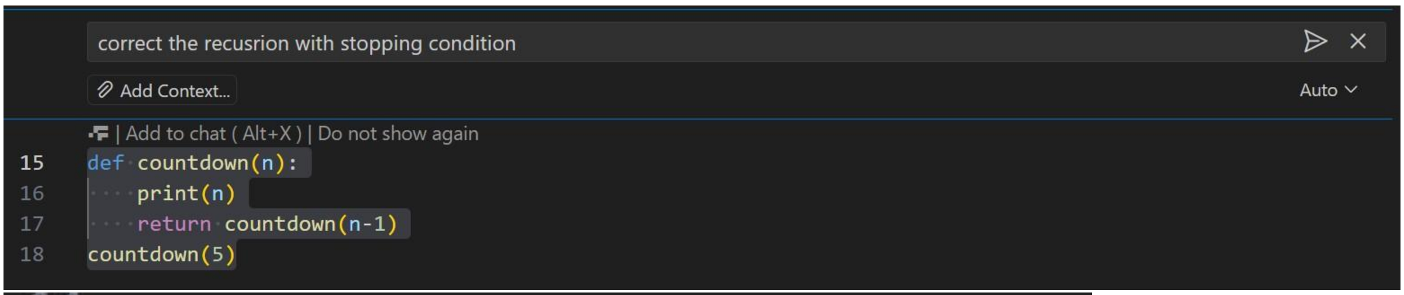
print(n) return

countdown(n-1)

countdown(5)

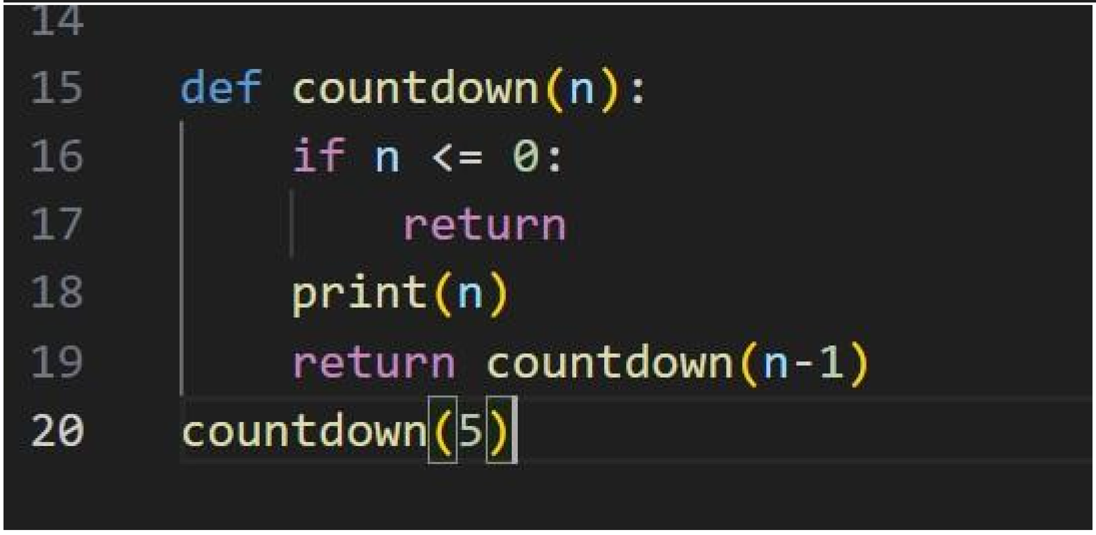
Expected Output : Correct recursion with stopping condition.

Screenshots:



correct the recusrion with stopping condition

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

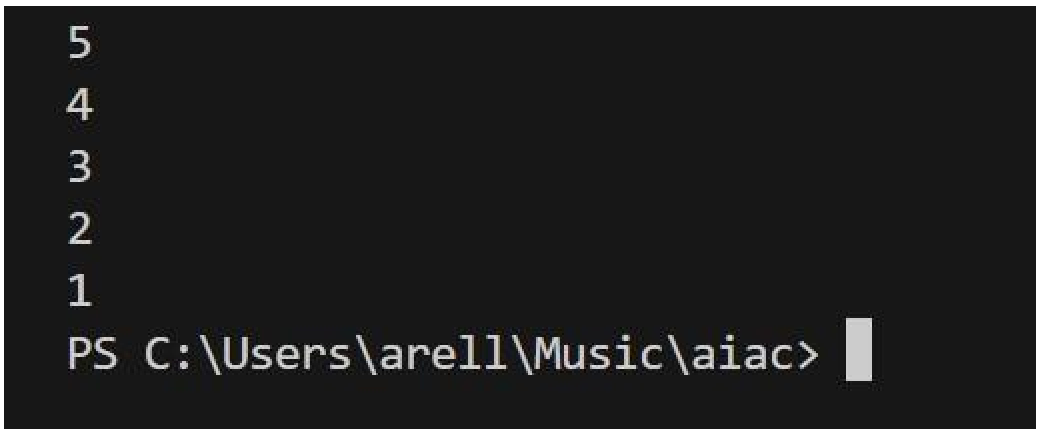


```
def countdown(n):
    if n <= 0:
        return
    print(n)
    return countdown(n-1)
countdown(5)
```

Code:

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

output:



```
5
4
3
2
1
PS C:\Users\arell\Music\aiac>
```

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

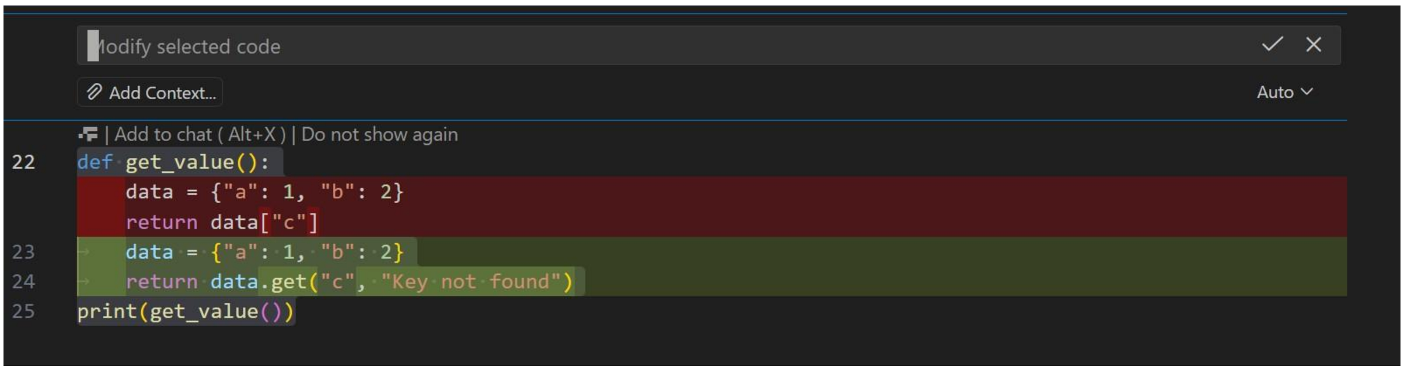
def get_value(): data = {"a": 1,

"b": 2} return data["c"]

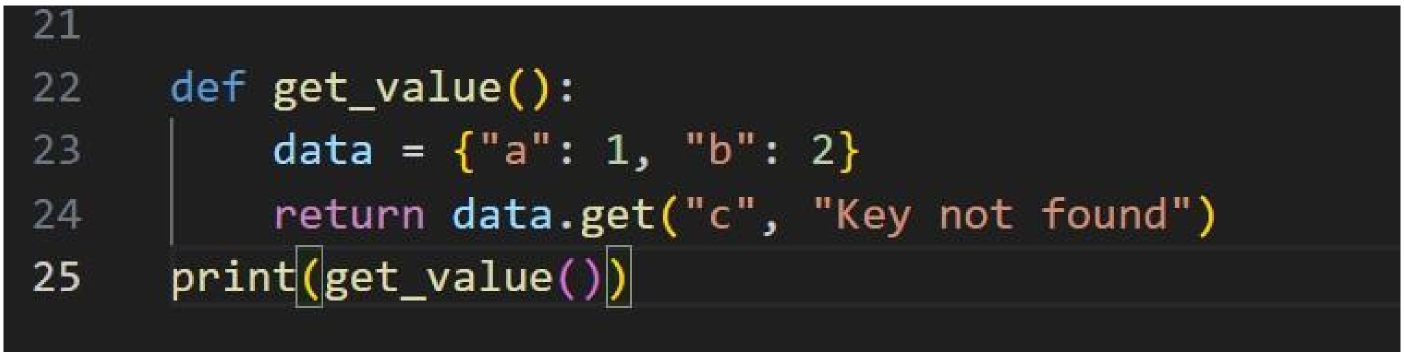
print(get_value())

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

Screenshots:



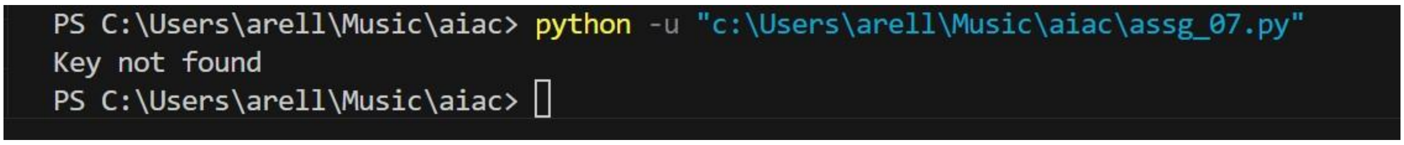
```
def get_value():
    data = {"a": 1, "b": 2}
    return data["c"]
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")
print(get_value())
```


```python

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

Code:

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

output:


```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
Key not found
PS C:\Users\arell\Music\aiac> 
```

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():

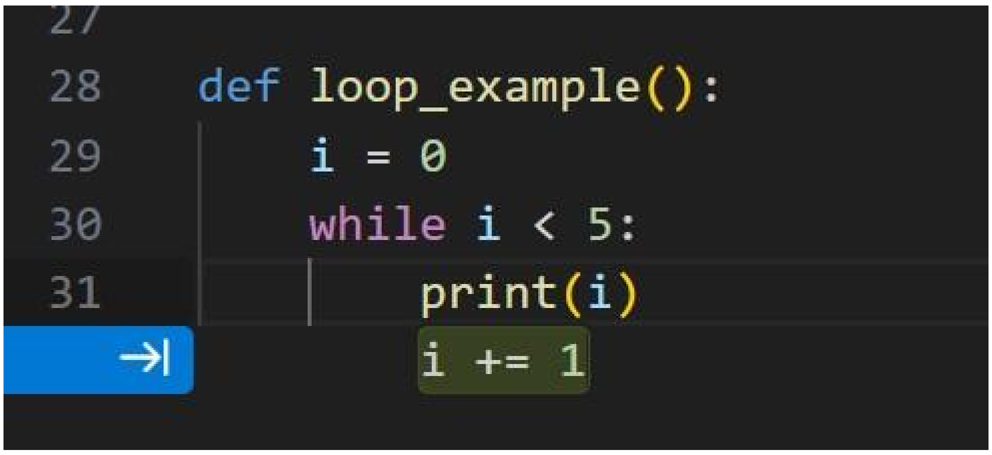
i = 0 while

i < 5:

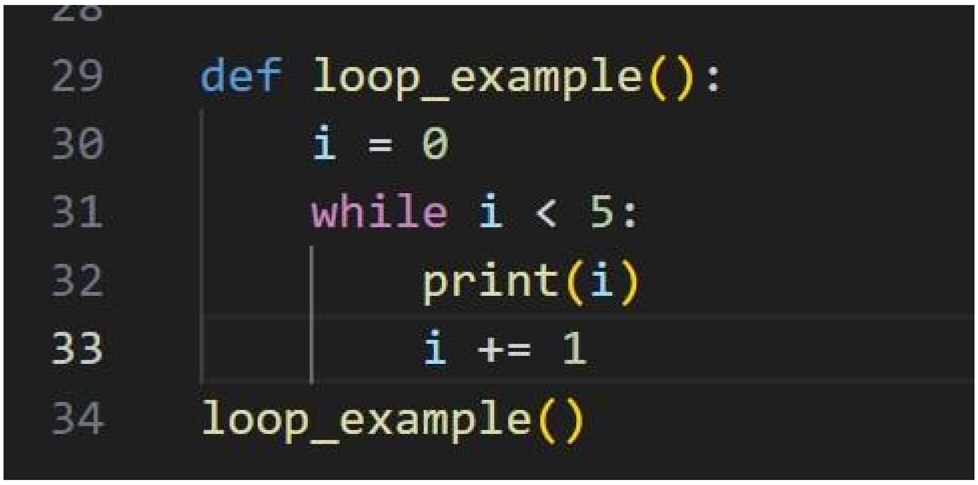
print(i)

Expected Output: Corrected loop increments i.

Screenshots:



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

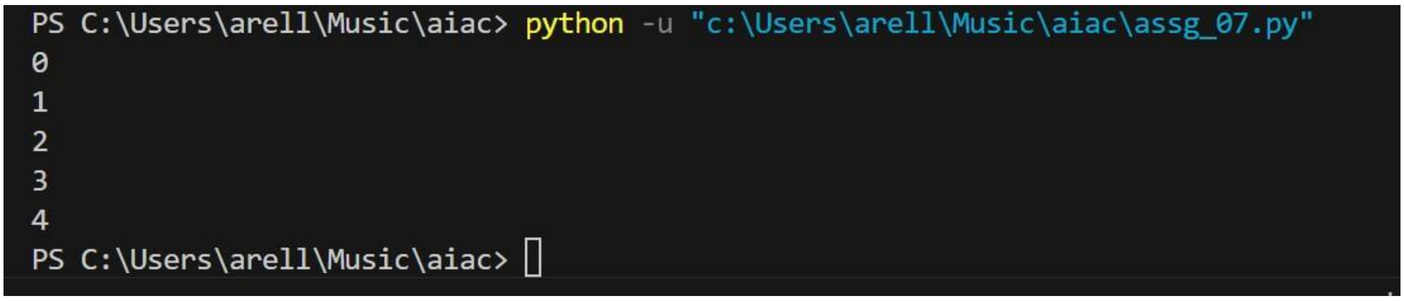


```
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1
loop_example()
```

Code:

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

output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
0
1
2
3
4
PS C:\Users\arell\Music\aiac> 
```

Task 6 (Unpacking Error – Wrong Variables)

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

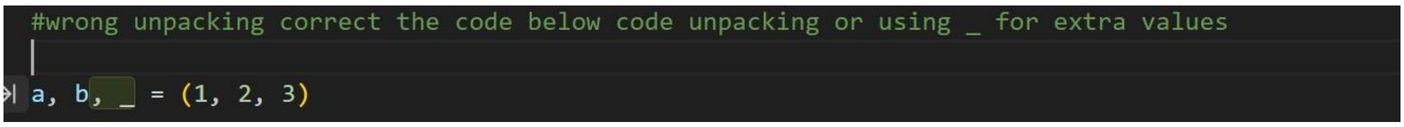
fix it.

Bug: Wrong unpacking

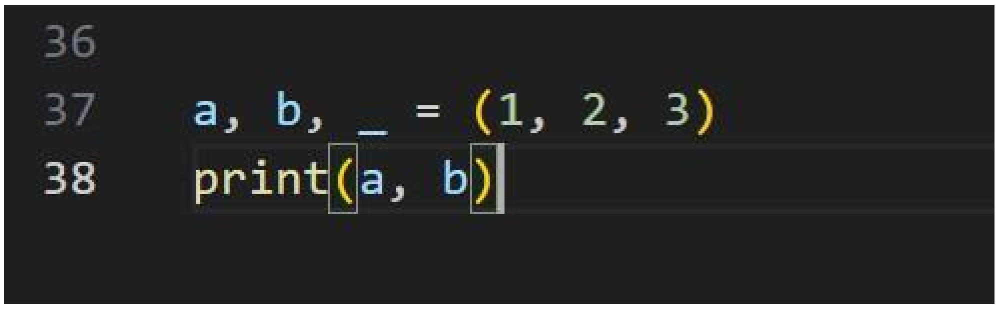
a, b = (1, 2, 3)

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

__Screenshots:__



```
#wrong unpacking correct the code below code unpacking or using _ for extra values

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

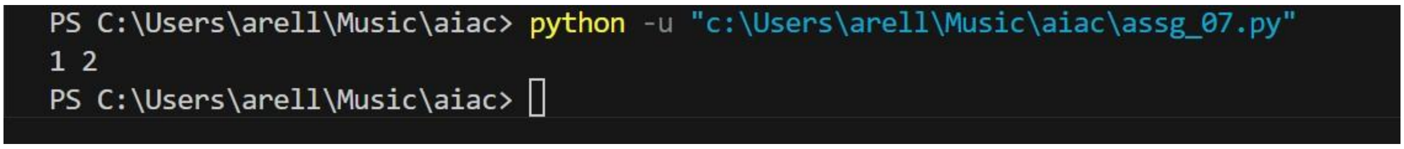


```
a, b, _ = (1, 2, 3)
print(a, b)
```

__Code:__

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

__output:__



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
1 2
PS C:\Users\arell\Music\aiac> 
```

__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():

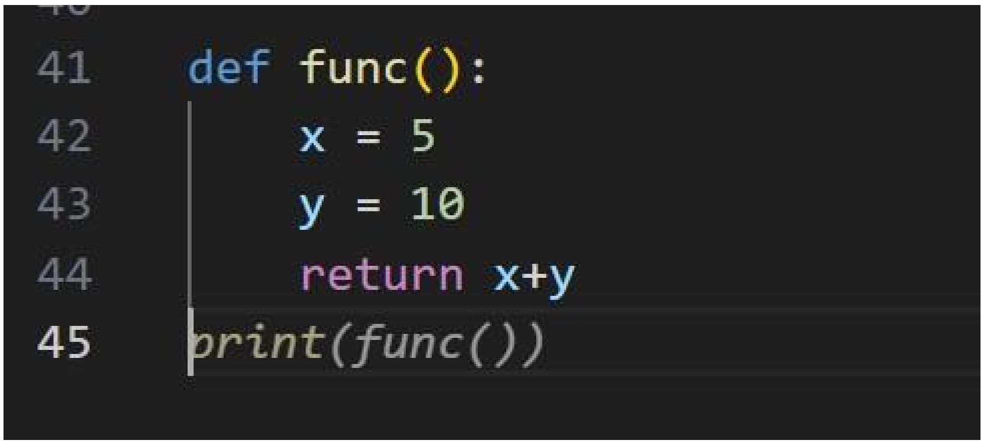
x = 5

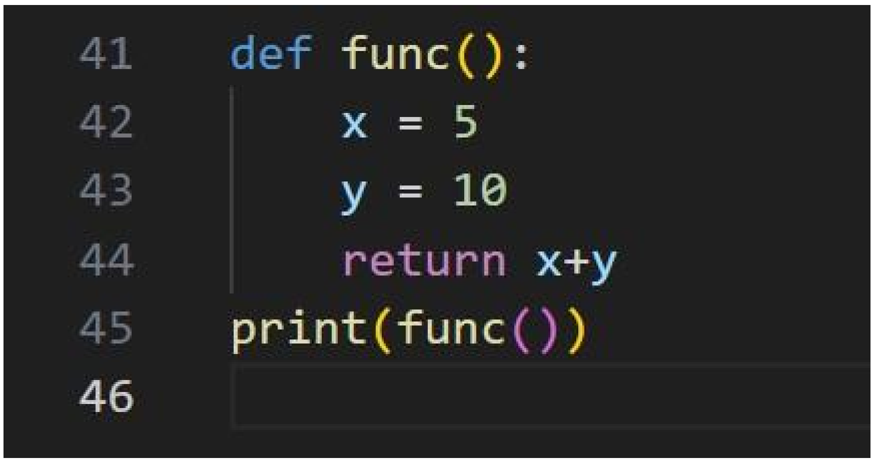
y = 10

return x+y

Expected Output : Consistent indentation applied.

__Screenshots__:

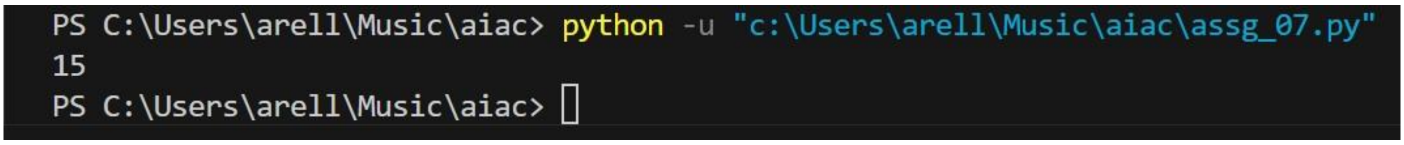

```
def func():
    x = 5
    y = 10
    return x+y
print(func())
```


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

Code:

```
def func():
x = 5      y =
10

    return x+y
print(func())
```

output:


```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
15
PS C:\Users\arell\Music\aiac> 
```

Task 8 (Import Error – Wrong Module Usage)

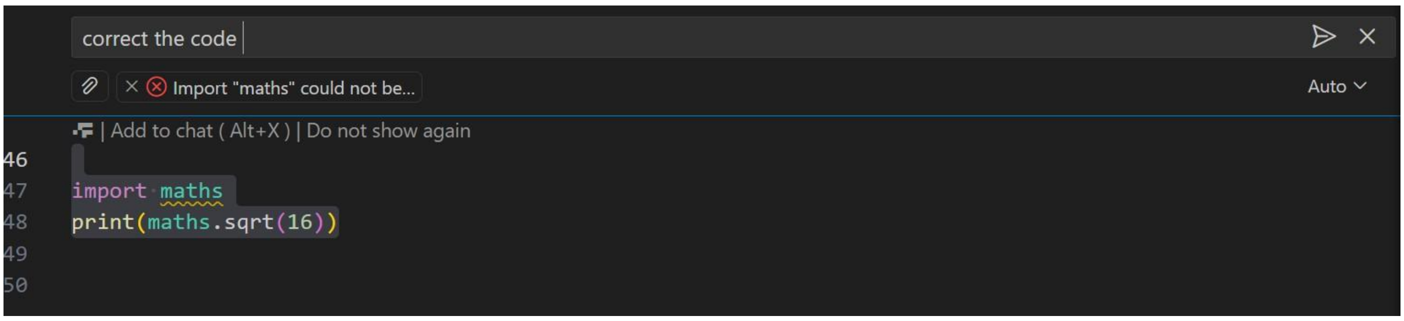
Task: Analyze given code with incorrect import. Use AI to fix.

Bug: Wrong import import

maths print(maths.sqrt(16))

Expected Output: Corrected to import math

Screenshots:

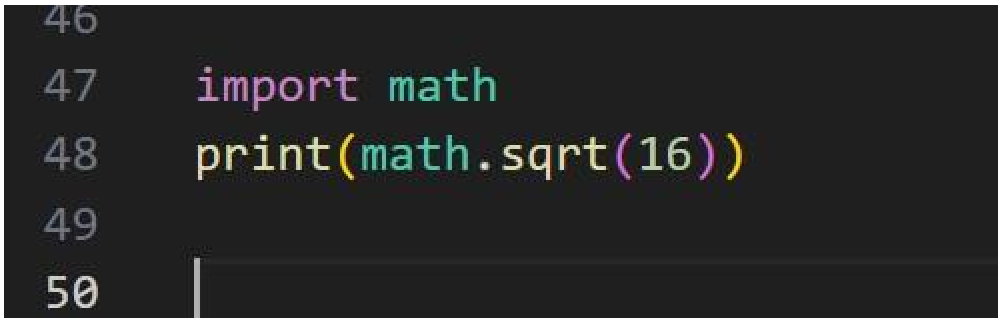

correct the code

Import "maths" could not be...

Auto

| Add to chat (Alt+X) | Do not show again

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

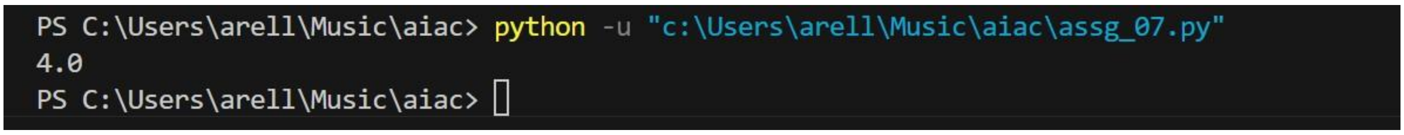


```
import math
print(math.sqrt(16))
```

Code:

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

output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
4.0
PS C:\Users\arell\Music\aiac> 
```

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:

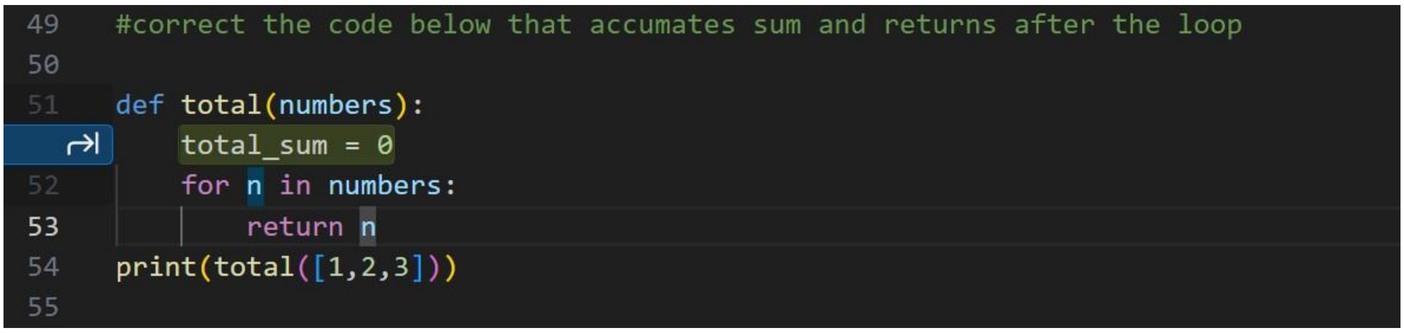
return n

print(total([1,2,3]))

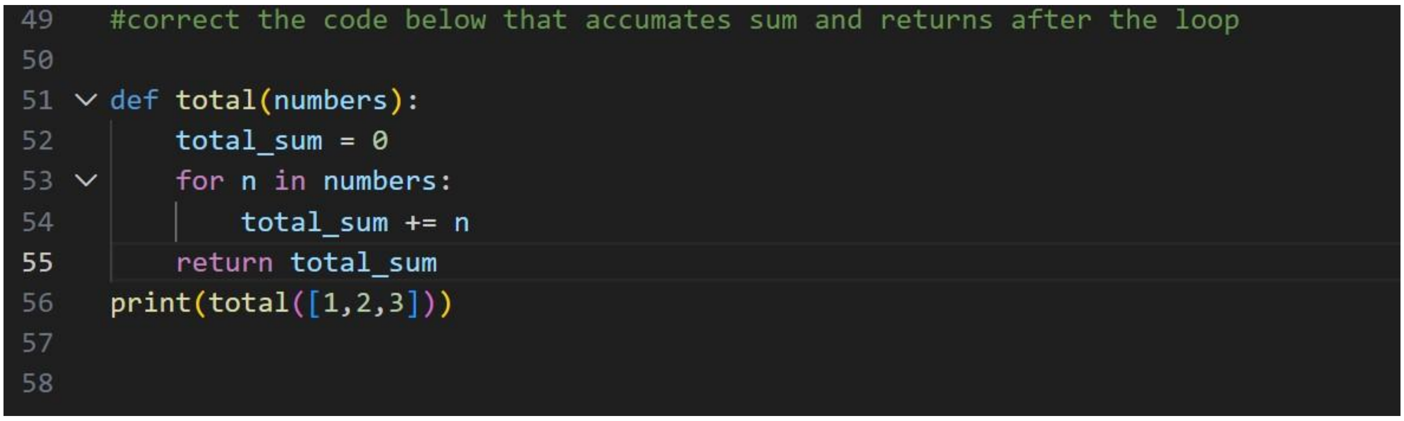
Expected Output: Corrected code accumulates sum and returns after

loop.

Screenshots:



```
#correct the code below that accumates sum and returns after the loop

def total(numbers):
    total_sum = 0
    for n in numbers:
        return n
print(total([1,2,3]))
```

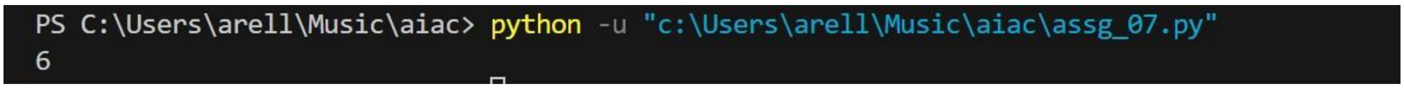


```
#correct the code below that accumates sum and returns after the loop

def total(numbers):
    total_sum = 0
    for n in numbers:
        total_sum += n
    return total_sum
print(total([1,2,3]))
```

Code:

```
def total(numbers):
total_sum = 0     for
n in numbers:
total_sum += n
return total_sum
print(total([1,2,3]))
```

output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
6
```

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.

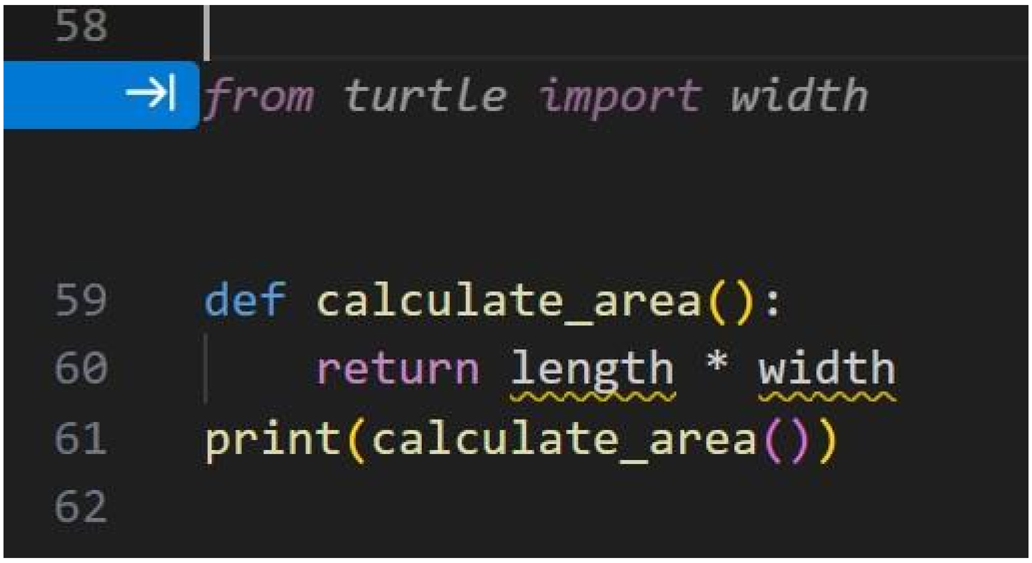
Expected Output :

• Corrected code with parameters.

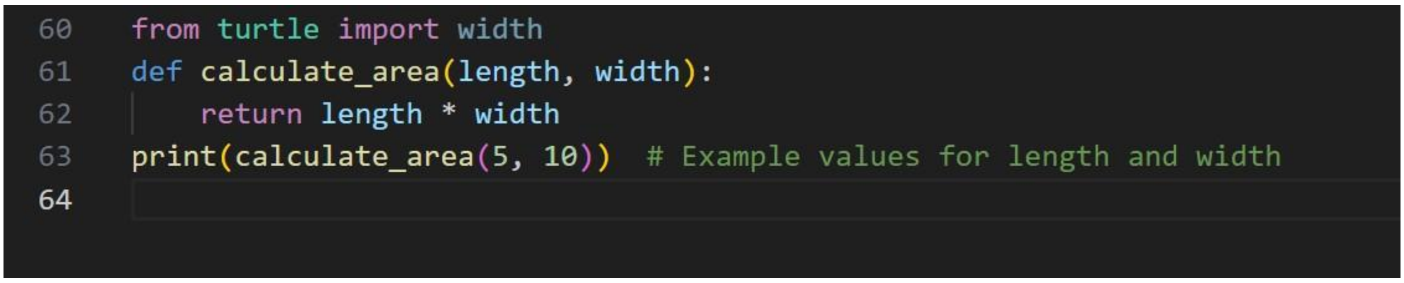
• AI explanation of the bug.

Successful execution of assertions.

Screenshots:



```
from turtle import width

def calculate_area():
    return length * width
print(calculate_area())
```

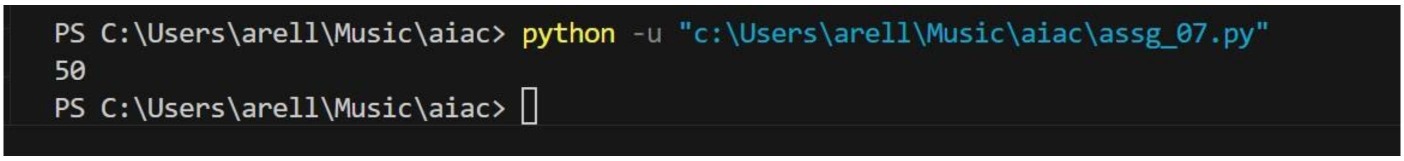


```
from turtle import width
def calculate_area(length, width):
    return length * width
print(calculate_area(5, 10))  # Example values for length and width
```

Code:

```
from turtle import width
def calculate_area(length, width):
    return length * width
print(calculate_area(5, 10))  # Example values for length and width
```

output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
50
PS C:\Users\arell\Music\aiac> 
```

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.

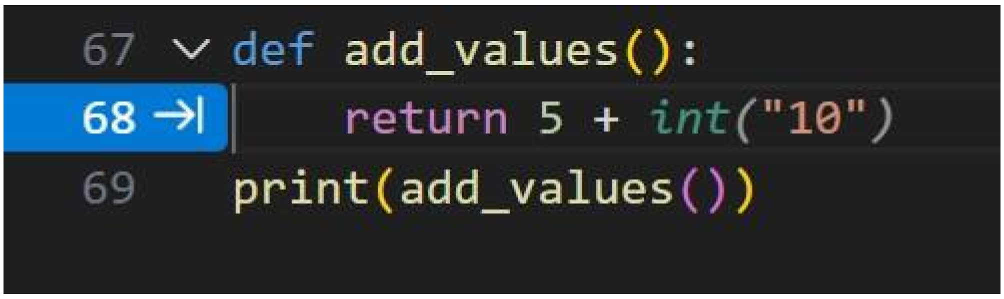
Expected Output #6:

• Corrected code with type handling.

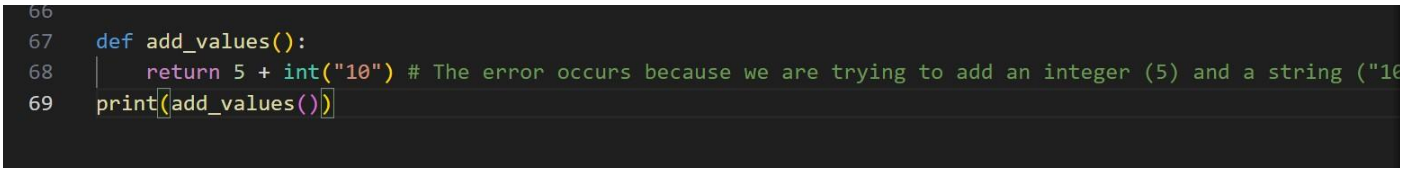
• AI explanation of the fix.

Successful test validation.

Screenshots:



```python
def add_values():
    return 5 + int("10")
print(add_values())
```



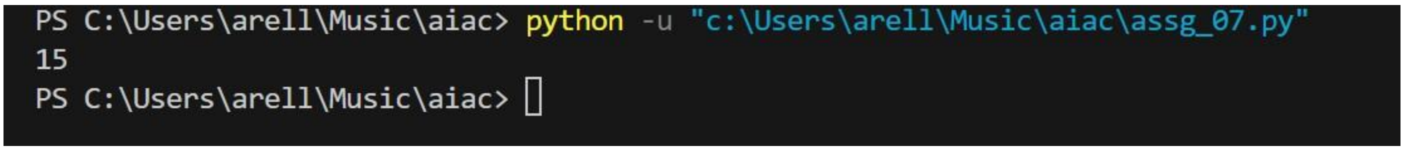
```python
def add_values():
    return 5 + int("10") # The error occurs because we are trying to add an integer (5) and a string ("1
print(add_values())
```

Code:

```python
def add_values():
    return 5 + int("10") # The error occurs because we are trying to add an
integer (5) and a string ("10"). To fix this, we need to convert the string
```

```
"10" to an integer using the int() function before performing the addition.
print(add_values())
```

output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
15
PS C:\Users\arell\Music\aiac> 
```

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.

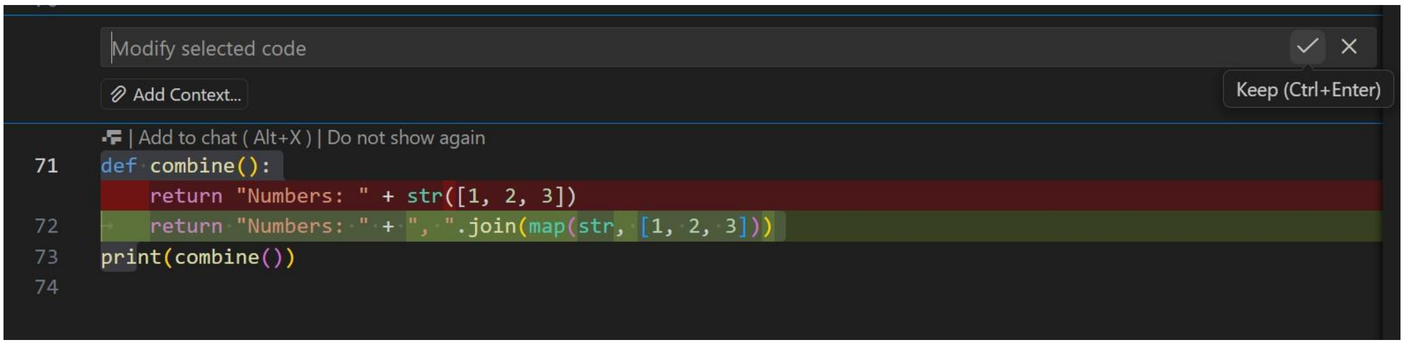
Expected Output:

• Corrected code

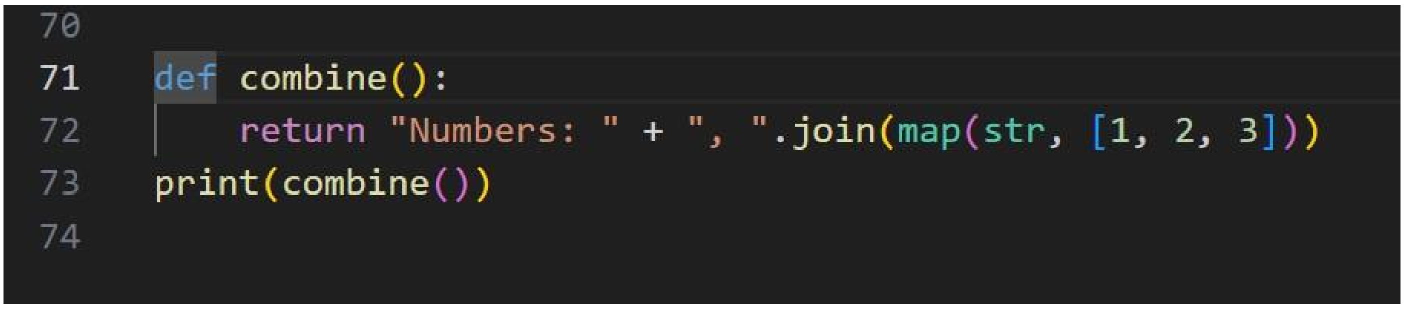
• Explanation

• Successful test validation

Screenshots:



```
def combine():
    return "Numbers: " + str([1, 2, 3])
    return "Numbers: " + ", ".join(map(str, [1, 2, 3]))
print(combine())
```

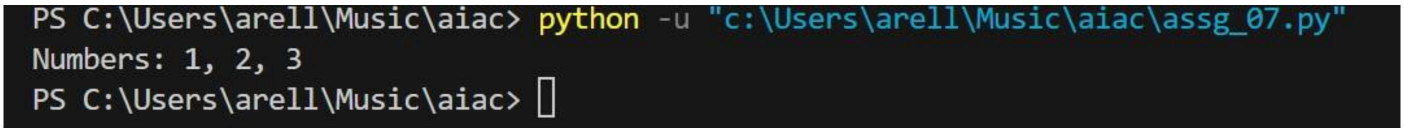


```python
def combine():
    return "Numbers: " + ", ".join(map(str, [1, 2, 3]))
print(combine())
```

code:

```
def combine():     return "Numbers: " + ",
".join(map(str, [1, 2, 3])) print(combine())
```

Output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\assg_07.py"
Numbers: 1, 2, 3
PS C:\Users\arell\Music\aiac> 
```

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:

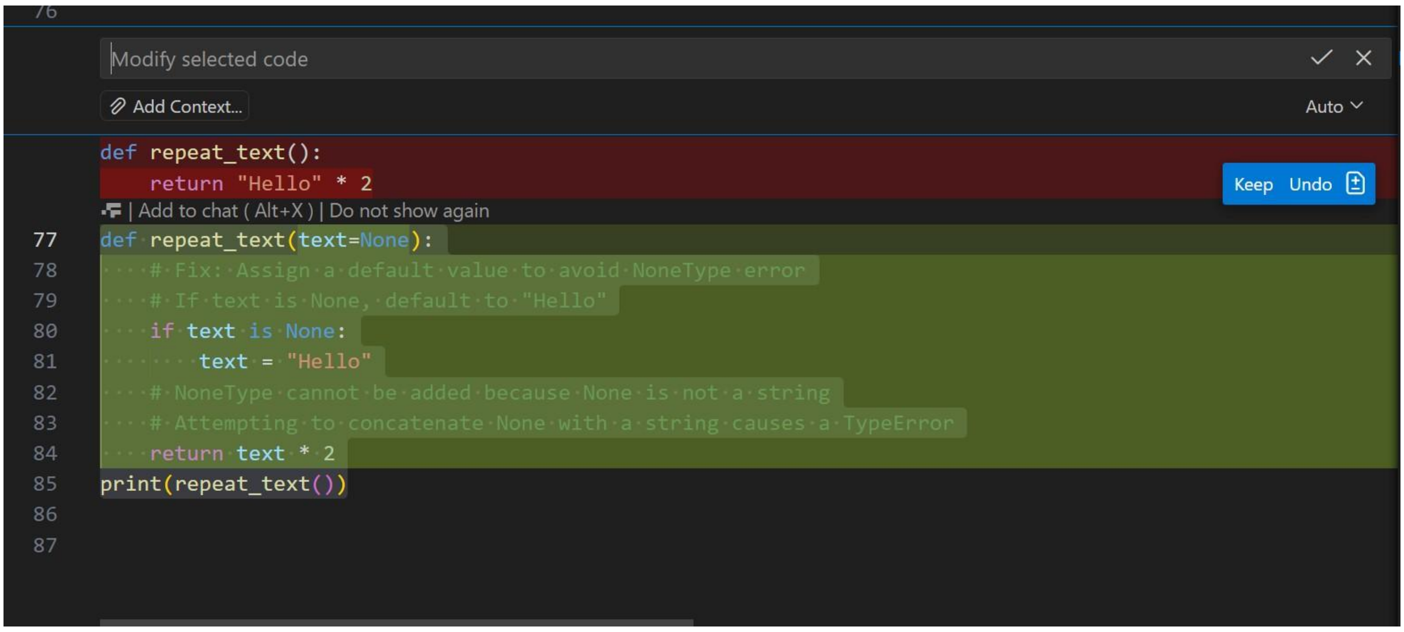
• Observe the error.

• Explain why float multiplication is invalid for strings.

• Fix by converting float to int.

• Add 3 assert test cases.

Screenshots:



Modify selected code

Add Context...

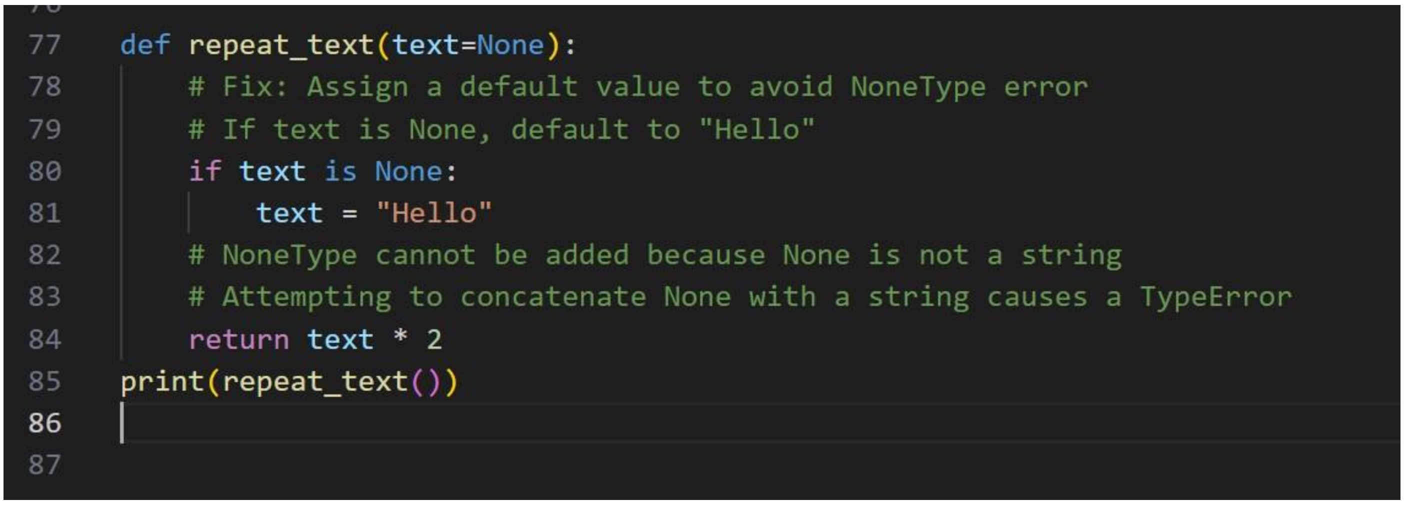
Auto

Keep Undo

```
def repeat_text():
    return "Hello" * 2
```

| Add to chat (Alt+X) | Do not show again

```
def repeat_text(text=None):
    # Fix: Assign a default value to avoid NoneType error
    # If text is None, default to "Hello"
    if text is None:
        text = "Hello"
    # NoneType cannot be added because None is not a string
    # Attempting to concatenate None with a string causes a TypeError
    return text * 2
print(repeat_text())
```

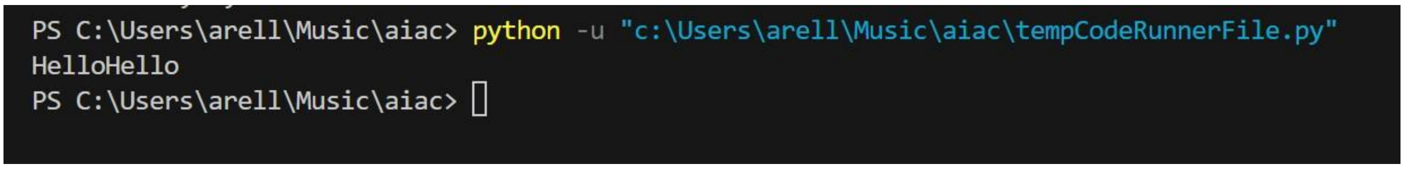


```
def repeat_text(text=None):
    # Fix: Assign a default value to avoid NoneType error
    # If text is None, default to "Hello"
    if text is None:
        text = "Hello"
    # NoneType cannot be added because None is not a string
    # Attempting to concatenate None with a string causes a TypeError
    return text * 2
print(repeat_text())
```

Code:

```
def repeat_text(text=None):
    # Fix: Assign a default value to avoid NoneType error
    # If text is None, default to
"Hello"     if text is None:         text
= "Hello"
    # NoneType cannot be added because None is not a string
    # Attempting to concatenate None with a string causes a TypeError
return text * 2 print(repeat_text())
```

output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\tempCodeRunnerFile.py"
HelloHello
PS C:\Users\arell\Music\aiac> 
```

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: ")

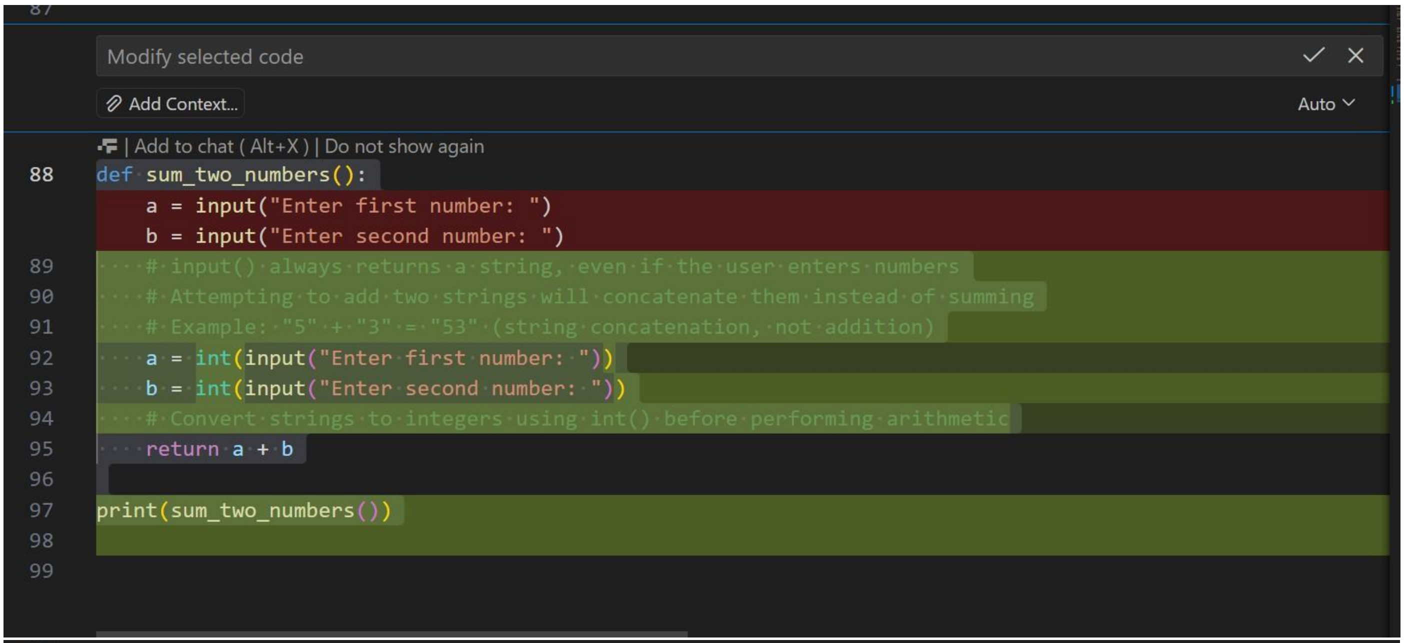
return a + b

print(sum_two_numbers())

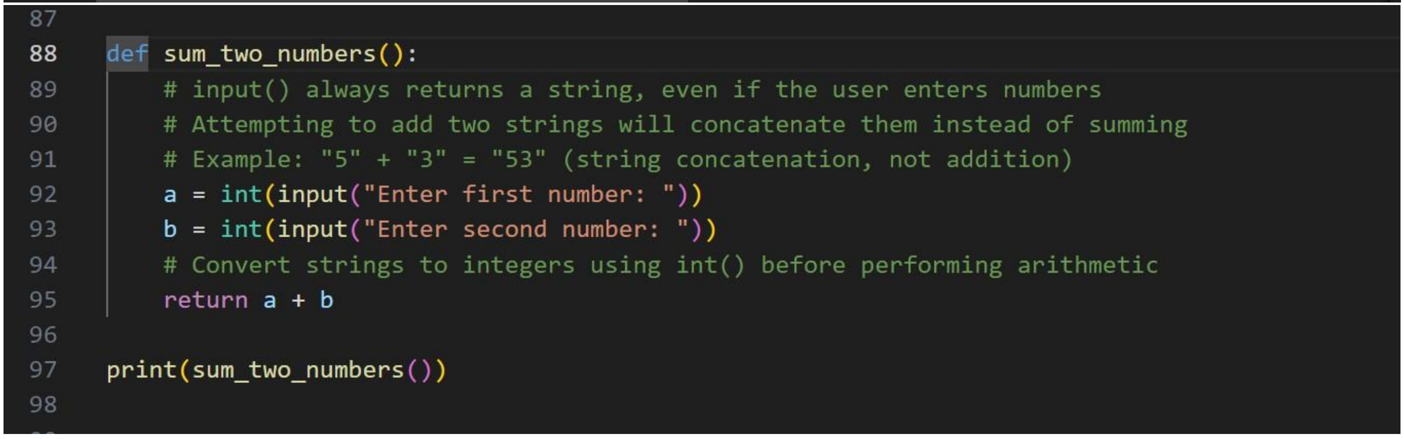
Requirements:

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

<u>Screenshots:</u>



```python
def sum_two_numbers():
    a = input("Enter first number: ")
    b = input("Enter second number: ")
    # input() always returns a string, even if the user enters numbers
    # Attempting to add two strings will concatenate them instead of summing
    # Example: "5" + "3" = "53" (string concatenation, not addition)
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    # Convert strings to integers using int() before performing arithmetic
    return a + b

print(sum_two_numbers())
```

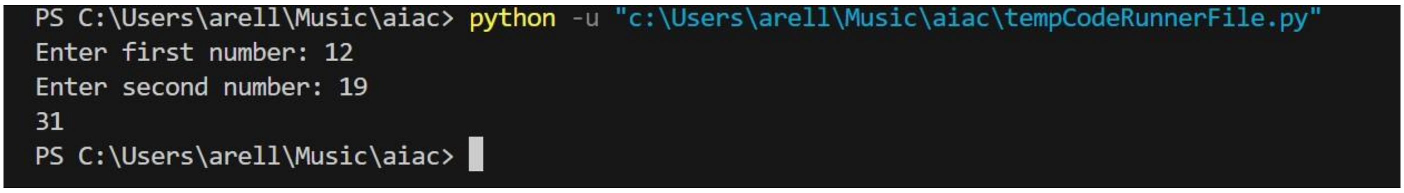


```python
def sum_two_numbers():
    # input() always returns a string, even if the user enters numbers
    # Attempting to add two strings will concatenate them instead of summing
    # Example: "5" + "3" = "53" (string concatenation, not addition)
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    # Convert strings to integers using int() before performing arithmetic
    return a + b

print(sum_two_numbers())
```

<u>Code:</u>

```python
def sum_two_numbers():
    # input() always returns a string, even if the user enters numbers
    # Attempting to add two strings will concatenate them instead of summing
    # Example: "5" + "3" = "53" (string concatenation, not addition)
a = int(input("Enter first number: "))     b = int(input("Enter
second number: "))
    # Convert strings to integers using int() before performing arithmetic
return a + b

print(sum_two_numbers())
```

output:



```
PS C:\Users\arell\Music\aiac> python -u "c:\Users\arell\Music\aiac\tempCodeRunnerFile.py"
Enter first number: 12
Enter second number: 19
31
PS C:\Users\arell\Music\aiac> 
```