Ai Assisted Coding

Week:7.5

Ht.No:2303A52376

Batch:45

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)
```

Prompt:

def add_item(item, items=[]):

items.append(item)

return items

print(add_item(1))

print(add_item(2) in this code there is a bug and it's a mutable default argument. Correct it.

Code:

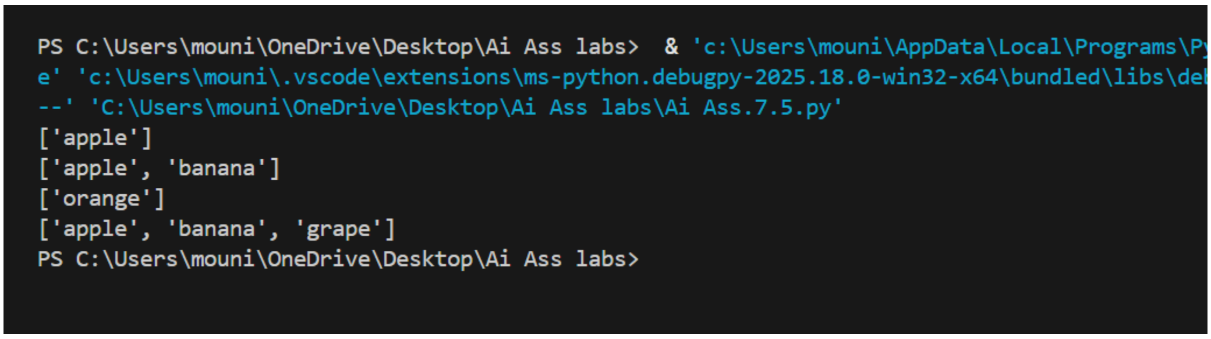
```
def add_item(item, items=[]):
    items.append(item)
    return items
print(add_item("apple"))  # Output: ['apple']
```

print(add_item("banana")) # Output: ['apple', 'banana']

print(add_item("orange", [])) # Output: ['orange']

print(add_item("grape")) # Output: ['apple', 'banana', 'grape']

Output:



```
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>  & 'c:\Users\mouni\AppData\Local\Programs\Py
e' 'c:\Users\mouni\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\deb
--' 'C:\Users\mouni\OneDrive\Desktop\Ai Ass labs\Ai Ass.7.5.py'
['apple']
['apple', 'banana']
['orange']
['apple', 'banana', 'grape']
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>
```

Explanation:

This code shows how a bug can be fixed in one line.

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

Prompt:

Analyze given code where floating-point comparison fails.correct with tolerance. Bug: Floating point precision issue

def check_sum():

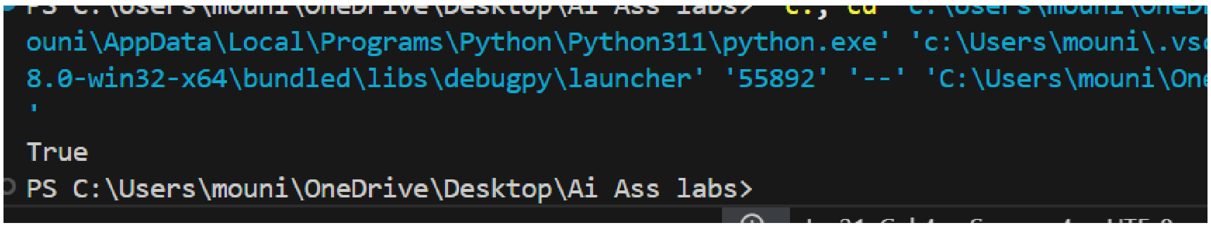
return (0.1 + 0.2) == 0.3

print(check_sum()) correct the code with tolerance level and precision recall.

Code:

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

Output:



```
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs> c:; cd 'c:\Users\mouni\OneDri
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mouni\.vs
8.0-win32-x64\bundled\libs\debugpy\launcher' '55892' '--' 'C:\Users\mouni\On
'
True
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>
```

Explanation:

The original function fails due to floating-point precision issues. The corrected function uses a tolerance level (1e-9) to compare the sum, ensuring that minor precision errors do not affect the outcome.

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)
```

Prompt:

Analyze given code where recursion runs infinitely due to missing base case. Use AI to fix.Bug: No base case

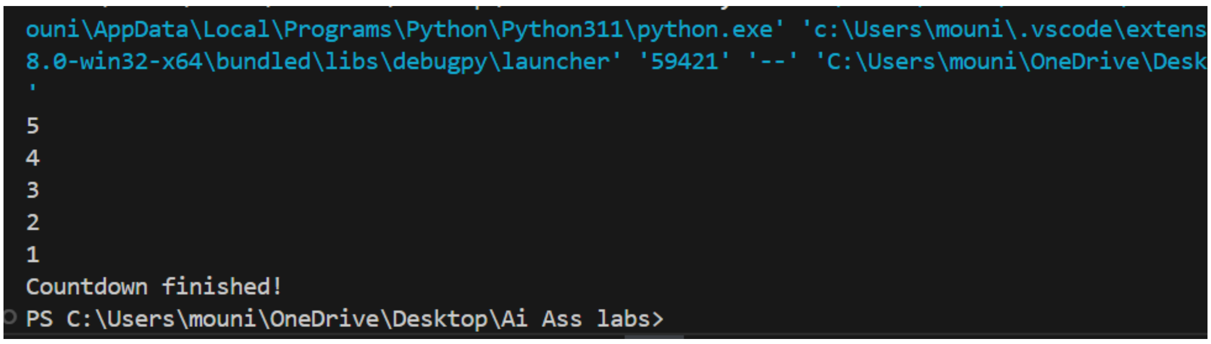
```
def countdown(n):
print(n)
return countdown(n-1)
```

countdown(5)

Code:

```
def countdown(n):
    if n <= 0:  # Base case to stop recursion
        print("Countdown finished!")
        return
    print(n)
    return countdown(n - 1)
countdown(5)
```

Output:



```
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mouni\.vscode\extens
8.0-win32-x64\bundled\libs\debugpy\launcher' '59421' '--' 'C:\Users\mouni\OneDrive\Desk
'
5
4
3
2
1
Countdown finished!
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>
```

Explanation:

The original code lacked a base case, causing it to call itself indefinitely. The corrected version includes a base case that checks if n is less than or equal to 0, at which point it prints a message and stops the 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())
```

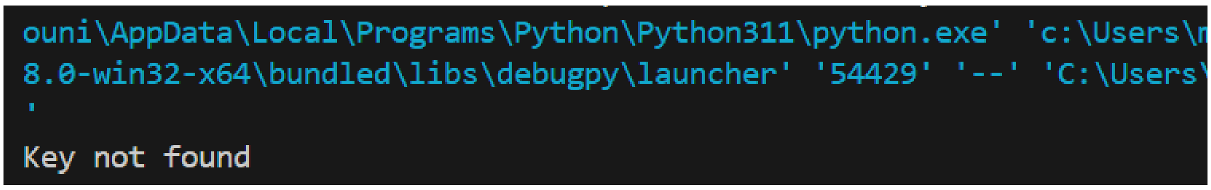
Prompt:

Generate a code for accessing non existing key,to fix a bug

Code:

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

Output:



```
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\
8.0-win32-x64\bundled\libs\debugpy\launcher' '54429' '--' 'C:\Users\
'
Key not found
```

Explanation: The corrected function uses the get() method with a default value to avoid KeyError.

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)
```

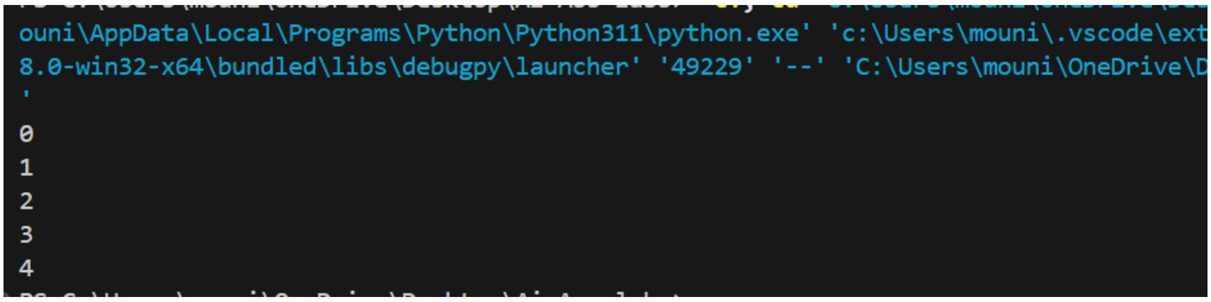
Prompt:

Generate a python code where the loop never ends and to fix the bug.

Code:

```
def loop_example():
    i = 0
    while i < 5:
        print(i)
        i += 1  # Increment to avoid infinite loop
loop_example()
```

Output:



```
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mouni\.vscode\ext
8.0-win32-x64\bundled\libs\debugpy\launcher' '49229' '--' 'C:\Users\mouni\OneDrive\D
'
0
1
2
3
4
```

Explanation:

The original code had an infinite loop because the variable 'i' was never incremented. By adding 'i += 1', we ensure that the loop will eventually terminate when 'i' reaches 5.

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)
```

Prompt:

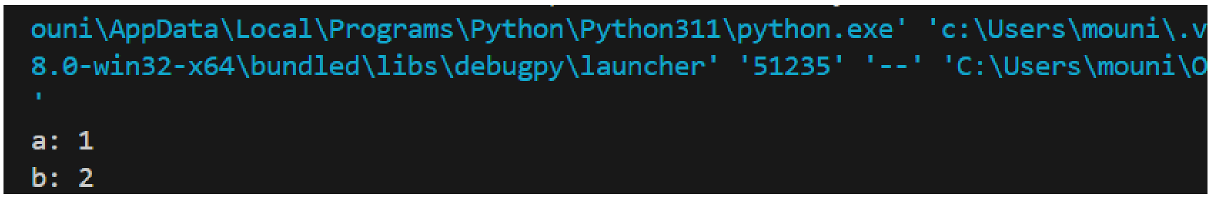
Generate a python code to fix a bug where tuple unpacking fails.

Code:

```
def unpack_example():
    a, b, _ = (1, 2, 3)  # Using _ to ignore extra value
    print("a:", a)
    print("b:", b)
```

.

Output:



```
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mouni\.v
8.0-win32-x64\bundled\libs\debugpy\launcher' '51235' '--' 'C:\Users\mouni\O
'
a: 1
b: 2
```

Explanation:

In the original code, there were three values in the tuple but only two variables to unpack into, which caused an error. By adding a third variable (using _), we can ignore the extra value and successfully unpack the first two values into a and b.

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
```

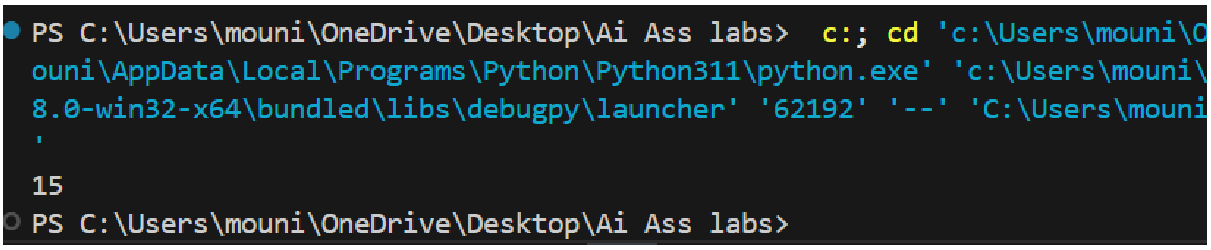
Prompt:

Generate a python code to fox the bug where mixed indentation breaks the execution.

Code:

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

Output:



```
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>  c:; cd 'c:\Users\mouni\O
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mouni\
8.0-win32-x64\bundled\libs\debugpy\launcher' '62192' '--' 'C:\Users\mouni
'
15
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>
```

Explanation:

In this code the print statement was not present.so without print statement we cant expext the output.

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))
```

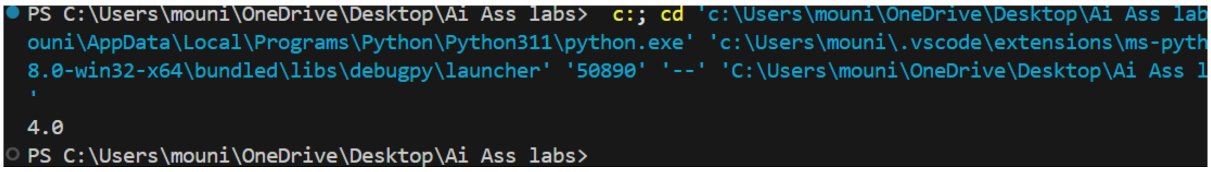
Prompt:

Generate a python code to fix the bug for a given code with correct import.

Code:

```
import math
def calculate_square_root(num):
    return math.sqrt(num)
print(calculate_square_root(16))
```

Output:



```
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>  c:; cd 'c:\Users\mouni\OneDrive\Desktop\Ai Ass lab
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mouni\.vscode\extensions\ms-pyth
8.0-win32-x64\bundled\libs\debugpy\launcher' '50890' '--' 'C:\Users\mouni\OneDrive\Desktop\Ai Ass l
'
4.0
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>
```

Explanation:

The original code had a typo in the module name ("maths" instead of "math"). The corrected code imports the correct module and defines a function to calculate the square root, which is then called with the argument 16.

Task:9.(Unreachable Code – Return Inside Loop) 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]))
```

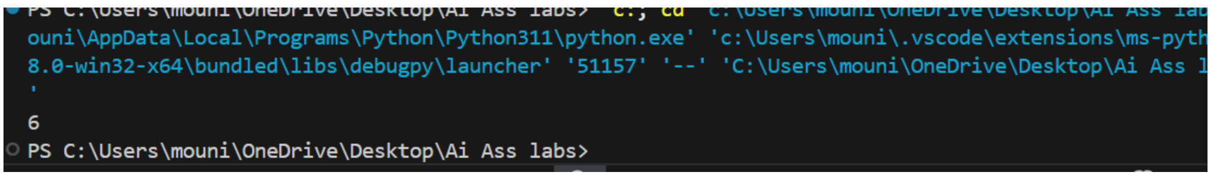
Prompt:

Generate a python code from unreachable code to reachable inside loop and prevents full iteration to fix a given bug.

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\mouni\OneDrive\Desktop\Ai Ass labs>  c:; cd  c:\Users\mouni\OneDrive\Desktop\Ai Ass lab
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mouni\.vscode\extensions\ms-pyth
8.0-win32-x64\bundled\libs\debugpy\launcher' '51157' '--' 'C:\Users\mouni\OneDrive\Desktop\Ai Ass l
'
6
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>
```

Explanation:

The original code had a return statement inside the loop, which caused it to return after the first iteration, making the rest of the numbers unreachable. The corrected code initializes a total_sum variable and accumulates the sum of all numbers in the loop, returning the final total after the loop completes.

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())
```

Prompt:

Generate a python code where a variable is used before being defined and fix the bug by defining length and width as parameters.Add 3 assert test cases for correctness.

Code:

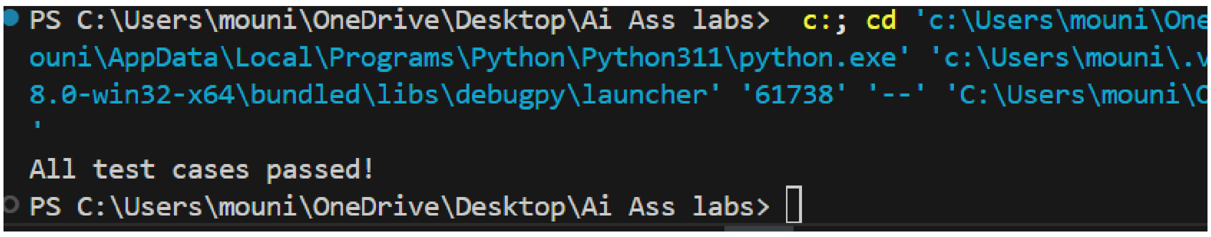
```
def calculate_area(length, width):
    return length * width
# Test cases
assert calculate_area(5, 10) == 50
```

assert calculate_area(7, 3) == 21

assert calculate_area(0, 5) == 0

print("All test cases passed!")

Output:



```
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs>  c:; cd 'c:\Users\mouni\One
ouni\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\mouni\.v
8.0-win32-x64\bundled\libs\debugpy\launcher' '61738' '--' 'C:\Users\mouni\C
'
All test cases passed!
PS C:\Users\mouni\OneDrive\Desktop\Ai Ass labs> 
```

Explanation:

The original code had a bug where the variables 'length' and 'width' were used before being defined, which would lead to a NameError. The corrected code defines 'length' and 'width' as parameters of the function 'calculate_area', allowing it to compute the area correctly. Additionally, three assert statements are included to test the correctness of the function with different inputs.