

## AIAC Assignment -7.5

Name: K.Lokesh

Ht.no:2303A51201

Bt.no:21

### Task 1 (Mutable Default Argument – Function Bug)

Task: Analyze given code where a mutable default argument cause unexpected behavior. Use AI to fix it.

# Bug: Mutable default argument

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

```
    items.append(item)
```

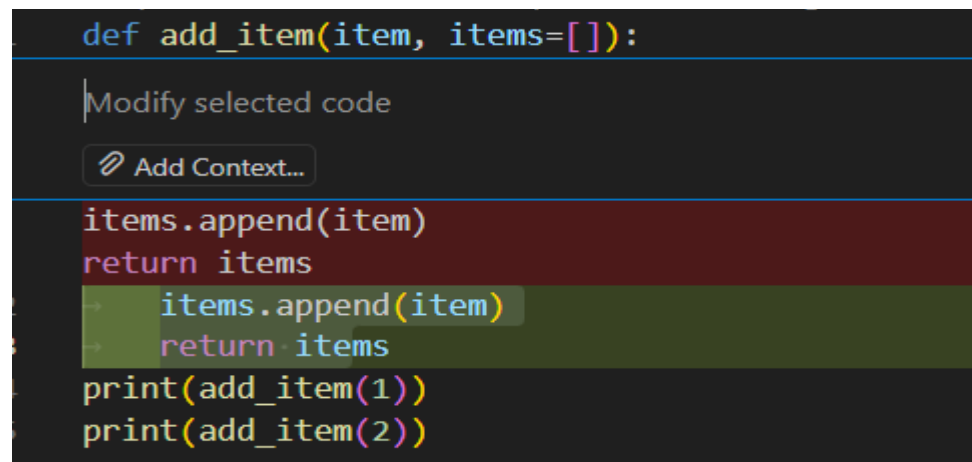
```
    return items
```

```
print(add_item(1))
```

```
print(add_item(2))
```

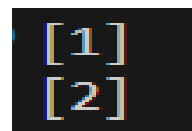
Expected Output: Corrected function avoids shared list bug.

Code:

A screenshot of a code editor with a dark background. The code is written in Python. The function definition is `def add_item(item, items=[]):`. Below it, there is a comment `# Modify selected code` and a button `Add Context...`. The function body contains `items.append(item)` and `return items`. Below the function definition, there are two print statements: `print(add_item(1))` and `print(add_item(2))`. The code is highlighted in a light green color.

```
def add_item(item, items=[]):  
    # Modify selected code  
    Add Context...  
    items.append(item)  
    return items  
print(add_item(1))  
print(add_item(2))
```

Output:

A screenshot of the output of the code. It shows two lines of output: `[1]` and `[2]`, each on a new line.

```
[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

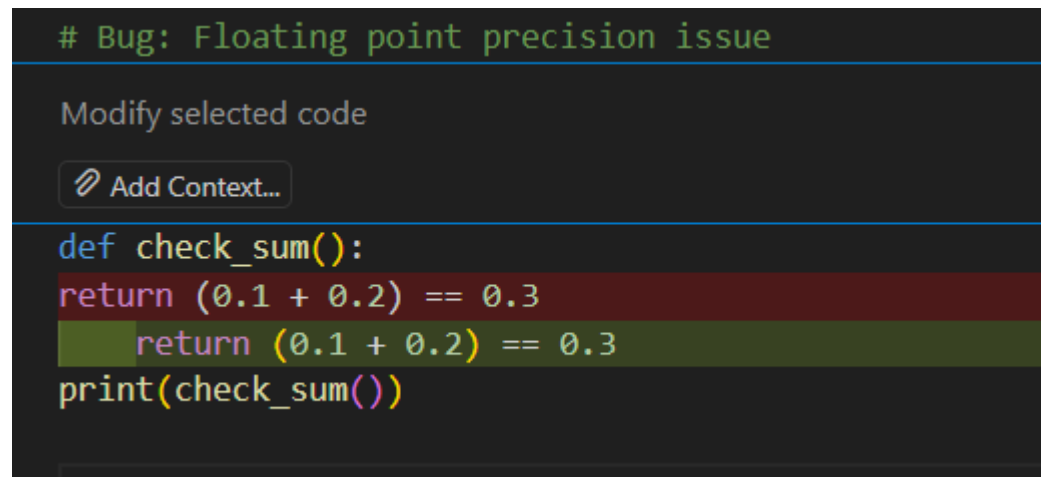
```
def check_sum():
```

```
    return (0.1 + 0.2) == 0.3
```

```
print(check_sum())
```

Expected Output: Corrected function

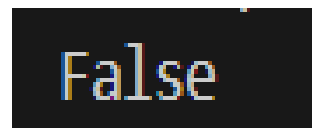
Code:



```
# Bug: Floating point precision issue

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

Output:



False

### 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.

Code:

```

1  # Bug: No base case
2
3  def countdown(n):
4      print(n)
5      return countdown(n-1)
6
7      if n < 0:
8          return
9          print(n)
10         return countdown(n-1)
11     countdown(5)

```

Modify selected code

Add Context...

Output:

```

5
4
3
2
1
0

```

#### 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:

```
# Bug: Accessing non-existing key

| Modify selected code

Add Context...

def get_value():
data = {"a": 1, "b": 2}
return data["c"]
→ data = {"a": 1, "b": 2}
→ return data.get("c", None)

print(get_value())
```

Output:

```
None
```

#### 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:

```
# Bug: Infinite loop

Modify selected code

Add Context...

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

### 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:

```
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.

Code:

```
def func():
    Modify selected code
    Add Context...
    x = 5
    y = 10
    return x+y
    result = func()print(result)
    x = 5
    y = 10
    return x+y
result = func()
print(result)
```

Output:

A terminal window with a black background showing the number 15 in a white, monospaced font.

#### Task 8 (Import Error – Wrong Module Usage)

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

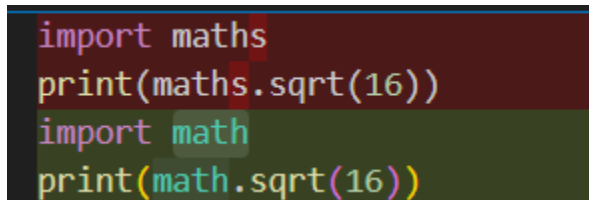
# Bug: Wrong import

```
import maths
```

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

Expected Output: Corrected to import math

Code:

A code editor with a dark background. The first two lines are highlighted in red: 'import maths' and 'print(maths.sqrt(16))'. The next two lines are highlighted in green: 'import math' and 'print(math.sqrt(16))'. This illustrates the correction of the import statement from 'maths' to 'math'.

Output:

A terminal window with a black background showing the number 4.0 in a white, monospaced font.