

Assignment -7.5

Name: Suhas

Ht.no:2303A51453

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:

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

Output:

```
[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:

The screenshot shows a code editor interface with a dark theme. At the top, there is a status bar with the text '# Bug: Floating point precision issue'. Below the status bar, the code is displayed. A tooltip 'Modify selected code' is shown above the code area. A button labeled 'Add Context...' is visible. The code itself is:

```
# Bug: Floating point precision issue

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

Output:

The screenshot shows a terminal window with a dark background. The word 'False' is printed in a light color, indicating the result of the floating-point comparison.

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     Modify selected code
4     ⚙ Add Context...
5
6 def countdown(n):
7     print(n)
8     return countdown(n-1)
9
10    if n < 0:
11        return
12    print(n)
13    return countdown(n-1)
14
15 countdown(5)
16
```

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 black rectangular box containing the number 15 in white.

Task 8 (Import Error – Wrong Module Usage)

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

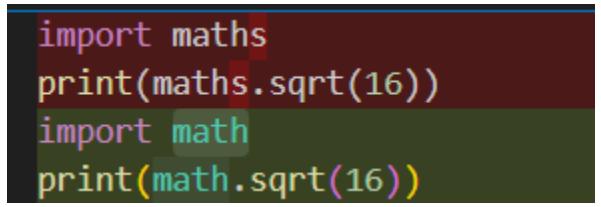
Bug: Wrong import

```
import maths
```

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

Expected Output: Corrected to import math

Code:

A code editor window showing four lines of Python code. The first line is 'import maths'. The second line is 'print(maths.sqrt(16))'. The third line is 'import math'. The fourth line is 'print(math.sqrt(16))'. The 'maths' module is highlighted in red, and the 'math' module is highlighted in green.

Output:

A black rectangular box containing the number 4.0 in white.