

## ASSIGNMENT 7.5

**NAME:** Varsha Sri Chukka

2303A52494

**BATCH:** 50

### 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.

```
#2303A52494
```

```
#Bug fix the code 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))'''
```

```
# fix the code without errors
```

```
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**

```
[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.

```
# Bug: Floating point precision issue  
'''def check_sum():  
    return (0.1 + 0.2) == 0.3  
    print(check_sum())'''  
# fix the code without errors  
def check_sum():  
    return abs((0.1 + 0.2) - 0.3) < 1e-9  
print(check_sum())
```

## OUTPUT

```
[1]  
[2]  
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.

```
# Bug: No base case
def countdown(n):
    if n == 0:
        return
    print(n)
    return 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.

```
# 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.

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

# Bug: Wrong unpacking

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

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

```
# Bug: Wrong unpacking
a, b = (1, 2, 3)
# fix the code without errors
a, b, c = (1, 2, 3)
```

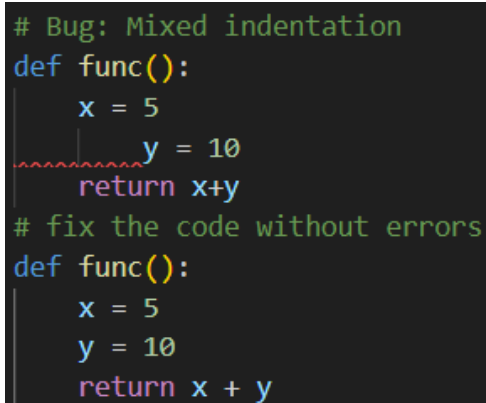
### Task 7 (Mixed Indentation – Tabs vs Spaces)

**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.



```
# Bug: Mixed indentation  
def func():  
    x = 5  
    y = 10  
    return x+y  
# fix the code without errors  
def func():  
    x = 5  
    y = 10  
    return x + y
```

### 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

```
# Bug: Wrong import
import maths
print(maths.sqrt(16))
# fix the code without errors
import math
print(math.sqrt(16))
```

### 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.

```
# Bug: Early return inside loop
'''def total(numbers):
    for n in numbers:
        return n
print(total([1,2,3]))'''
# fix the code without errors
def total(numbers):
    sum_total = 0
    for n in numbers:
        sum_total += n
    return sum_total
print(total([1,2,3]))
```

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.

```
# Bug: Using undefined variable
'''def calculate_area():
    return length * width
print(calculate_area())'''
# fix the code without errors
def calculate_area(length, width):
    return length * width
print(calculate_area(5, 10))
```

50

### 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.

```
# Bug: Adding integer and string
def add_values():
    return 5 + "10"
print(add_values())
# fix the code without errors
def add_values():
    return 5 + int("10")
print(add_values())
```

## OUTPUT

```
15
```

### 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



```
# Bug: Adding string and list
def combine():
    return "Numbers: " + [1, 2, 3]
print(combine())
# fix the code without errors
def combine():
    return "Numbers: " + str([1, 2, 3])
print(combine())
```

```
Numbers: [1, 2, 3]
```

### 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.

```
# Bug: Multiplying string by float
def repeat_text():
    return "Hello" * 2.5
print(repeat_text())
# fix the code without errors
def repeat_text():
    return "Hello" * int(2.5)
print(repeat_text())
```

### Task 14 (Type Error – Adding None to Integer)

Task: Analyze code where None is added to an integer.

# Bug: Adding None and integer

```
def compute():  
    value = None  
    return value + 10  
  
print(compute())
```

Requirements:

- Run and identify the error.
- Explain why NoneType cannot be added.
- Fix by assigning a default value.
- Validate using asserts.

```
# Bug: Adding None and integer  
def compute():  
    value = None  
    return value + 10  
print(compute())  
# fix the code without errors  
def compute():  
    value = 0  
    return value + 10  
print(compute())
```

10

### 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.

```
# 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())

# fix the code without errors
def sum_two_numbers():
    a = float(input("Enter first number: "))
    b = float(input("Enter second number: "))
    return a + b

print(sum_two_numbers())
```

```
Enter first number: 2.3
Enter second number: 6.1
8.399999999999999
```