AI Assisted Coding

ASSIGNMENT 7.4

Name: Madadi Uday

HT No: 2303A52038

Batch: 31

Question 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.

Prompt:

Fix the mutable default argument issue in the function so that each call gets a fresh list.

Code:

Output(O/P)

Explanation:

The default list [] was shared between calls.

Default arguments are created only once.
Using None ensures a new list is created each time.

Question 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:

Fix floating point comparison using tolerance.

Code:

Output(O/P)

Explanation:

Floating numbers are not exact in binary.

Instead of ==, we compare the difference with a small tolerance.

Question 3: Recursion Error – Missing Base Case

Task:

Analyze given code where recursion runs infinitely due to missing base case. Use AI to fix it.

```
# Bug: No base case
def countdown(n):
    print(n)
    return countdown(n-1)
countdown(5)
```

Expected Output : Correct recursion with stopping condition.

Prompt:

Add a stopping condition to prevent infinite recursion.

Code:

Output(O/P)

Explanation:

Without a base case, recursion never stops.

The base case stops when n becomes 0.

Question 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.

Prompt:

Fix missing key access safely.

Code:

Output(O/P)

Explanation:

Accessing data["c"] causes KeyError.
.get() prevents crashes and allows default value.

Question 5: Infinite Loop – Wrong Condition

Task:

Analyze given code where the 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.

Prompt:

Fix the infinite loop by incrementing the counter.

Code:

Output(O/P)

Explanation:

‘i’ was never incremented.

The loop condition stayed True forever.

Question 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.

Prompt:

Fix tuple unpacking.

Code:

Output(O/P)

Explanation:

Tuple had three values but only two variables.

Added c to Fix.

Question 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.

Prompt:

Fix inconsistent indentation.

Code:

Output(O/P)

Explanation:

Python requires consistent indentation.
Using spaces consistently fixes execution.

Question 8: Import Error – Wrong Module Usage

Task:

Analyze given code with incorrect import. Use AI to fix it.

Bug: Wrong import

import maths

print(maths.sqrt(16))

Expected Output: Corrected to import math

Prompt:

Fix incorrect module import.

Code:

Output(O/P)

Explanation:

Corrected Typo of Import of math module

Question 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

Prompt:

Fix early return inside loop.

Code:

Output(O/P)

Explanation:

Return statement inside the loop stops after the first iteration.

Now it sums all elements before returning.

Question 10: Name Error – Undefined Variable

Task:

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.

Prompt:

Fix undefined variables by using parameters. Add 3 assert test cases to test correctness.

Code:

Output(O/P)

Explanation:

length and width were undefined.
Making them parameters fixes NameError.

Question 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.

Prompt:

Fix int adding string error using conversion. Add 3 assert test cases to test correctness, explain in a docstring why int + str is invalid.

Code:

Output(O/P)

Explanation:

Python cannot add int and str directly.

Type conversion solves the issue.

Question 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

Prompt:

Fix string + list concatenation using conversion, Add 3 assert test cases to test correctness, explain in a docstring why str + list is invalid.

Code:

Output(O/P)

Explanation:

String cannot be directly added to the list.

Convert list to string first.

Question 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.

Prompt:

Fix string multiplied by float using conversion, Explain in a docstring why float multiplication is invalid for strings, Add 3 assert test cases.

Code:

Output(O/P)

Explanation:

Strings can only be multiplied by integers.
Convert float to int before multiplication.

Question 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.

Prompt:

Fix None addition issue, Explain in a docstring why NoneType cannot be added, Validate using asserts.

Code:

Output(O/P)

Explanation:

NoneType cannot be added to int.
Assign default numeric value before operation.

Question 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.

Prompt:

Fix input conversion issue using type conversion, Explain in docstring why input is always string, Verify with assert test cases.

Code:

Output(O/P)

Explanation:

input() always returns string.
Convert to float or int before addition.