

## ASSIGNMENT-7.5

Name: D.Sandeep

HT NO: 2303A51665

Batch: 23

### 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.

The screenshot shows a terminal window with the following content:

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1  def add_item(item,items=None):
2      if items is None:
3          items=[]
4      items.append(item)
5      print(items)
6  add_item(3)
7  add_item(1)
8
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
```

The terminal output is:

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
[3]
[1]
PS D:\AI>
```

### 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

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1 import math
2 def check_sum():
3     return math.isclose(0.1 + 0.2, 0.3)
4 print(check_sum())

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
True
3
2
1
0
PS D:\AI>
```

### 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.

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1 def countdown(n):
2     if n<0:
3         return
4     print(n)
5     return countdown(n-1)
6 countdown(3)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
not found
0
1
2
3
4
1 2
```

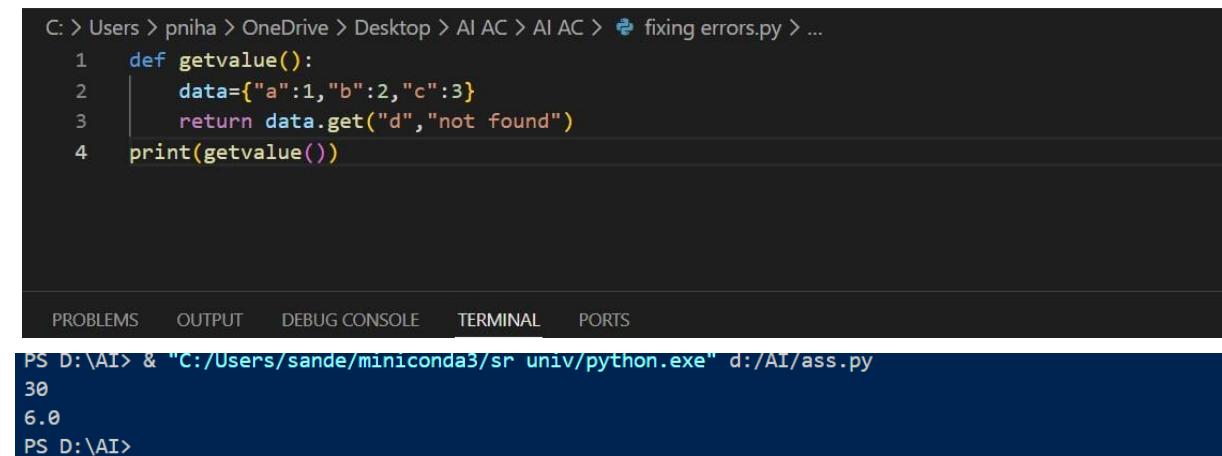
#### Task 4 (Dictionary Key Error)

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.



The screenshot shows a code editor interface with a dark theme. At the top, there's a navigation bar with tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is currently selected. Below the tabs, the code editor displays the following Python code:

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...  
1 def getvalue():  
2     data={"a":1,"b":2,"c":3}  
3     return data.get("d", "not found")  
4 print(getvalue())
```

Below the code editor, the terminal window shows the execution of the script:

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py  
30  
6.0  
PS D:\AI>
```

#### 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.

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
0
1
2
3
4
PS D:\AI>
```

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1  def loopexample():
2      i=0
3      while i<5:
4          print(i)
5          i+=1
6  loopexample()
7
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

### 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.

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1  a, b, _ = (1, 2, 3)
2  print(a, b)
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
1 2
PS D:\AI>
```

### 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.

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1 def function():
2     x=10
3     y=20
4     return x+y
5 print(function())

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
30
PS D:\AI>
```

### 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

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py
1 import math
2 print(math.sqrt(36))

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
6.0
PS D:\AI>
```

### 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.

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1 def total(numbers):
2     sum=0
3     for i in numbers:
4         sum+=i
5     return sum
6 print(total([1,2,3]))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
6
PS D:\AI>
```

### 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.

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1 #function to calculate the area of a rectangle
2 def calculate_area(length, width):
3     #multiply length and width to get the area
4     return length * width
5     #call the function with example values
6 length = 5
7 width = 3
8 area = calculate_area(length, width)
9 print(f"The area of the rectangle is: {area}")

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
The area of the rectangle is: 15
PS D:\AI>
```

### 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.

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1  def add_values():
2      #indent the return statement to be inside the function
3      #convert the string inputs to integers before adding
4      return 5+int("10")
5      #call the function and print the result
6  print(add_values())
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
15
PS D:\AI>
```

### 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

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1  # str + list is invalid because Python cannot concatenate a string with a list directly
2  # strings and lists are different types, and the + operator doesn't know how to combine them
3  # You must convert the list to a string first using str() or join()
4  def combine():
5      # Fix: Convert list to string using str()
6      return "Numbers: " + str([1, 2, 3])
7  print(combine())
8  # verify with 3 assert cases
9  assert combine() == "Numbers: [1, 2, 3]", "Test 1 failed"
10 assert isinstance(combine(), str), "Test 2 failed"
11 assert "Numbers:" in combine(), "Test 3 failed"
12 print("All assertions passed!")
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
Numbers: [1, 2, 3]
All assertions passed!
PS D:\AI>
```

### 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

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1  # str * float is invalid because Python cannot multiply a string by a float
2  # The * operator for strings only works with integers to repeat the string
3  # You must convert the float to an integer first using int()
4  def repeat_text():
5      # Fix: Convert float to int
6      return "Hello" * int(2.5)
7  print(repeat_text())
8  # Verify with 3 assert cases
9  assert repeat_text() == "HelloHello", "Test 1 failed"
10 assert isinstance(repeat_text(), str), "Test 2 failed"
11 assert len(repeat_text()) == 10, "Test 3 failed"
12 print("All assertions passed!")
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py
HelloHello
All assertions passed!
```

### 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 = Nonereturn 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.

The screenshot shows the VS Code interface. The top part displays a Python script named 'fixing errors.py' with the following code:

```
C: > Users > pniha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...  
1  def compute():  
2      value = 0 # Assign a default value  
3      return value + 10  
4  result = compute()  
5  print(result)  
6  # Validate using asserts  
7  assert result == 10, "Test 1 failed"  
8  assert isinstance(result, int), "Test 2 failed"  
9  assert result > 0, "Test 3 failed"  
10 print("All assertions passed!")
```

The bottom part shows the terminal output:

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/ass.py  
10  
All assertions passed!
```

### 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.

```
C:\> Users > pninha > OneDrive > Desktop > AI AC > AI AC > fixing errors.py > ...
1 def sum_two_numbers():
2     a = int(input("Enter first number: ")) # Convert input to int
3     b = int(input("Enter second number: ")) # Convert input to int
4     return a + b
5 result = sum_two_numbers()
6 print(result)
7 # Verify with assert test cases
8 assert isinstance(result, int), "Result should be an integer"
9 assert result == (int(input("Enter first number: ")) + int(input("Enter second number: "))), "Sum does not match expected value"
10 print("All assertions passed!")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
PS D:\AI> & "C:/Users/sande/miniconda3/bin/univ/python.exe" d:/AI/ass.py
Enter first number: 11
Enter second number: 18
29
Enter first number: 
```