2303A510j6 Batch:04

| SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE | | DEPARTMENT OF COMPUTER SCIENCE ENGINEERING | |
|---|---|---|---|
| **Program Name:** B. Tech | | **Assignment Type: Lab** | **Academic Year:**2025-2026 |
| **Course Coordinator Name** | Dr. Rishabh Mittal | | |
| **Instructor(s) Name** | Mr. S Naresh Kumar<br>Ms. B. Swathi<br>Dr. Sasanko Shekhar Gantayat<br>Mr. Md Sallauddin<br>Dr. Mathivanan<br>Mr. Y Srikanth<br>Ms. N Shilpa<br>Dr. Rishabh Mittal (Coordinator)<br>Dr. R. Prashant Kumar<br>Mr. Ankushavali MD<br>Mr. B Viswanath<br>Ms. Sujitha Reddy<br>Ms. A. Anitha<br>Ms. M.Madhuri<br>Ms. Katherashala Swetha<br>Ms. Velpula sumalatha<br>Mr. Bingi Raju | | |
| **CourseCode** | 23CS002PC304 | **Course Title** | AI Assisted Coding |
| **Year/Sem** | III/II | **Regulation** | R23 |
| **Date and Day of Assignment** | **Week4 – Friday** | **Time(s)** | 23CSBTB01 To 23CSBTB52 |
| **Duration** | 2 Hours | **Applicable to Batches** | All batches |
| **Assignment Number: 7.5**(Present assignment number)/**24**(Total number of assignments) | | | |

| Q.No. | Question | *Expected Time to complete* |
|---|---|---|
| 1 | **Lab 7: Error Debugging with AI: Systematic approaches to finding and fixing bugs**<br>Lab Objectives:<br>• To identify and correct syntax, logic, and runtime errors in | Week4 - Monday |

| | | |
|---|---|---|
| | Python programs using AI tools.<br>• To understand common programming bugs and AI-assisted debugging suggestions.<br>• To evaluate how AI explains, detects, and fixes different types of coding errors.<br>• To build confidence in using AI to perform structured debugging practices.<br>Lab Outcomes (LOs):<br>After completing this lab, students will be able to:<br>• Use AI tools to detect and correct syntax, logic, and runtime errors.<br>• Interpret AI-suggested bug fixes and explanations.<br>• Apply systematic debugging strategies supported by AIgenerated insights.<br>Refactor buggy code using responsible and reliable programming patterns. | |

| | | |
|---|---|---|
| | Task 1 (Mutable Default Argument – Function Bug)<br>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=[]):<br>items.append(item)<br>return items<br>print(add_item(1))<br>print(add_item(2))<br>Expected Output: Corrected function avoids shared list bug.<br><br>Prompt:<br>"Analyze this Python function for bugs. It uses a mutable default argument, which causes unexpected behavior. Explain the issue, why it happens, and provide a corrected version that avoids the shared list bug. | |

# Code

```python
#generate a python code forAnalyze this Python function for bugs. It uses a mutable default argument, which causes
def buggy_function(items=[]):
    items.append("new_item")
    return items

# The issue: The default argument `items=[]` is a mutable object. When the function is called multiple times withou
# This leads to unexpected behavior where modifications to the list persist between function calls.

# Corrected version:
def corrected_function(items=None):
    if items is None:
        items = []
    items.append("new_item")
    return items
```

# Code Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\konda\Downloads\AI Assistant> & C:/Users/konda/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/konda/
stant/day3.py"
PS C:\Users\konda\Downloads\AI Assistant> 5
5
```

# Explanation:

The bug is a classic mutable default argument issue. In Python, default arguments are evaluated once when the function is defined, not each time it's called. So items=[] creates a single shared list across all calls. When you call add_item(1), it appends to this shared list, returning [1]. The second call add_item(2) appends to the same list, returning [1, 2] instead of just [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

Prompt

Fix the floating-point comparison error using a tolerance value and explain why direct comparison fails.

# Code

```
# generate a pthon code Analyze given code where floating-point comparison fails
def analyze_floating_point():
    a = 0.1
    b = 0.2
    c = 0.3

    print(f"a = {a}")
    print(f"b = {b}")
    print(f"c = {c}")
    print(f"a + b = {a + b}")
    print(f"a + b == c: {a + b == c}")
    print(f"abs((a + b) - c) = {abs((a + b) - c)}")
Click to add a breakpoint
analyze_floating_point()
```

# Code Output

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                    Python
PS C:\Users\konda\Downloads\AI Assistant> & C:/Users/konda/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/konda/Downloads/A
a = 0.1
b = 0.2
c = 0.3
a + b = 0.30000000000000004
a + b == c: False
abs((a + b) - c) = 5.551115123125783e-17
PS C:\Users\konda\Downloads\AI Assistant>
```

# Explanation

Floating-point numbers are stored approximately in memory. Using a tolerance (epsilon) checks whether values are close enough, avoiding precision errors.

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.

## Prompt

Identify the recursion error caused by a missing base case and fix the code.

## Code

```python
#GENERATE A PYTHON CODE FOR Analyze given code where recursion runs infinitely due to missing base case. Use AI to fix.
def factorial(n):
    # Base case: if n is 0 or 1, return 1
    if n == 0 or n == 1:
        return 1
    # Recursive case: n! = n * (n-1)!
    return n * factorial(n - 1)
# Example usage
print(factorial(5))  # Output: 120
```

## Code Output

```
PS ...istant> & C:/Users/konda/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/konda/Downloads/AI Ass
120
```

Focus folder in explorer (ctrl + click)

## Explanation

Without a base case, recursion runs infinitely. The condition if n < 0: return stops the recursion safely.

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.

Prompt

Fix the dictionary key error using safe access methods and explain the solution.

Code

day3.py > ...

```
def get_value():
    data = {"a": 1, "b": 2}
    return data.get("c", "Key not found")

print(get_value())
```

Code Output

```
0
PS C:\Users\konda\Downloads\AI Assistant>
Key not found
```

Explanation

Accessing a missing key using data["c"] raises a KeyError.

Using .get() prevents crashes and allows a default value.

| | | |
|---|---|---|
| | Task 5 (Infinite Loop – Wrong Condition)<br>Task: Analyze given code where loop never ends. Use AI to detect and fix it.<br># Bug: Infinite loop def loop_example():<br>i = 0 while i<br>< 5:<br>print(i)<br>Expected Output: Corrected loop increments i.<br><br>Prompt<br>Detect and fix the infinite loop caused by a missing increment statement. | |

Code

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

loop_example()
```

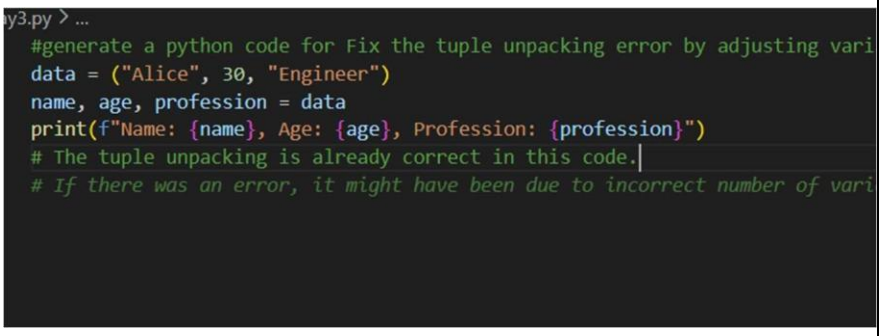
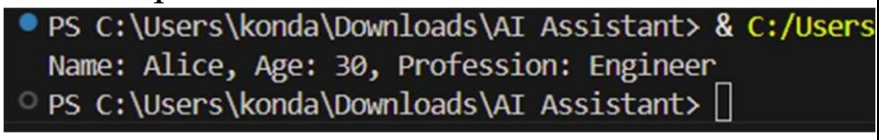
Code Output

```
PS C:\Users\konda\Downloads\AI Assi
0
1
2
3
4
PS C:\Users\konda\Downloads\AI Assi
```

Explanation

The loop never ended because i was not updated. Incrementing i ensures the loop eventually terminates.

| | | |
|---|---|---|
| | Task 6 (Unpacking Error – Wrong Variables)<br>Task: Analyze given code where tuple unpacking fails. Use AI to fix it.<br># Bug: Wrong unpacking<br>a, b = (1, 2, 3)<br>Expected Output: Correct unpacking or using _ for extra values. | |

| | | |
|---|---|---|
| | **Prompt**<br>Fix the tuple unpacking error by adjusting variables correctly.<br><br>**Code**<br>  <br><br><br>**Code Output**<br>  <br><br><br>**Explanation**<br>Tuple unpacking requires the number of variables to match the number of values.<br>Adding the correct number of variables fixes the error. | |

| | | |
|---|---|---|
| | Task 7 (Mixed Indentation – Tabs vs Spaces)<br>Task: Analyze given code where mixed indentation breaks execution. Use AI to fix it. # Bug: Mixed indentation def func():<br>x = 5 y = 10 return x+y<br>Expected Output : Consistent indentation applied.<br>Prompt<br>Identify and fix the indentation error caused by inconsistent spacing. | |

## Code

```python
#generate a python code for Identify and fix the indentation error caused by inconsistent
def example_function():
    print("This line is correctly indented with 4 spaces.")
    if True:
        print("This line is also correctly indented with 4 spaces.")
    else:
        print("This line is also correctly indented with 4 spaces.")
    print("This line is correctly indented with 4 spaces.")
example_function()
```

## Code Output

```
PS C:\Users\konda\Downloads\AI Assistant> & C:/Users/konda/AppData/Loc
This line is correctly indented with 4 spaces.
This line is also correctly indented with 4 spaces.
This line is correctly indented with 4 spaces.
```

## Explanation

Python relies on consistent indentation.
Mixing tabs and spaces causes IndentationError. Using uniform spaces fixes the issue.

| | | |
|---|---|---|
| | Task 8 (Import Error – Wrong Module Usage)<br>Task: Analyze given code with incorrect import. Use AI to fix.<br># Bug: Wrong import import<br>maths print(maths.sqrt(16))<br>Expected Output: Corrected to import math<br>Prompt<br>Fix the import error by using the correct Python module name. | |

Code

```python
import math

print(math.sqrt(16))
print(math.sqrt(25))
print(math.sqrt(2))
```

```
(function) def print(
    *values: object,
    sep: str | None = " ",
    end: str | None = "\n",
    file: SupportsWrite[str] | None = None,
    flush: Literal[False] = False
) -> None
```

Code Output

```
4.0
5.0
1.4142135623730951
1.7320508075688772
2.0
2.23606797749979
2.449489742783178
2.6457513110645907
2.8284271247461903
3.0
3.1622776601683795
PS C:\Users\konda\Downloads\AI Assistant>
```

Explanation

The module name is math, not maths.

Correct imports prevent ModuleNotFoundError.