

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: M.Tech. and MCA		Assignment Type: Lab	
Course Coordinator Name		Venkataramana Veeramsetty	
Course Code		Course Title	AI Assisted Problem Solving Using Python
Year/Sem	I/I	Regulation	R24
Date and Day of Assignment	Week3 - Monday	Time(s)	
Duration	2 Hours	Applicable to Batches	M.Tech. and MCA
AssignmentNumber: 7.3(Present assignment number)/24(Total number of assignments)			
ASSIGNMENT – 7			
Name: Rimsha Mujeeb		Roll Number:2503B05138 (M.Tech - CSE)	

Q.No.	Question
Q1.	<p>Task Description#1</p> <ul style="list-style-type: none"> Paste a function with a missing colon (add(a, b)), and let AI fix the syntax error. <pre>python def add(a, b) return a + b</pre> <p>Expected Output#1</p> <ul style="list-style-type: none"> Corrected function with syntax fix

Q2.

Task Description#2 (Loops)

- Identify and fix a logic error in a loop that causes infinite iteration.

```
python

def count_down(n):
    while n >= 0:
        print(n)
        n += 1 # Should be n -= 1
```

Expected Output#2

- AI fixes increment/decrement error

The screenshot shows a Google Colab notebook titled "AIPP_ASSIGNMENT-7.ipynb". The code cell contains the following Python function:

```
def count_down(n):
    while n >= 0:
        print(n)
        n += 1 # Should be n -= 1
```

The AI has provided a detailed explanation of the error:

If `n` starts at, say, 5, it will become 6, then 7, and so on. Since `n` is always increasing, it will always be greater than or equal to 0, making the `while` condition perpetually `True`. This results in an `infinite loop`.

The AI has suggested a fix:

To correct this, we need to **decrement** `n` in each iteration, typically by `n -= 1` (or `n = n - 1`). This ensures that `n` will eventually become less than 0, causing the `while` loop condition (`n >= 0`) to become `False` and the loop to terminate. Additionally, we might want to include a base case or a stopping condition to make sure the loop actually counts down to a specific value and then stops.

The corrected code is shown in the cell:

```
def count_down_fixed(n):
    while n >= 0:
        print(n)
        n -= 1 # Corrected: Decrement n instead of incrementing

# Let's test the corrected function
print("\n--- Testing the corrected count_down_fixed function ---")
count_down_fixed(5)
print("Count down complete!")
```

The output of the corrected function is displayed below the code cell:

```
--- Testing the corrected count_down_fixed function ---
5
4
3
2
1
0
Count down complete!
```

The notebook interface includes tabs for "Variables" and "Terminal", and a status bar at the bottom showing the date and time.

Q3.

Task Description#3

- Debug a runtime error caused by division by zero. Let AI insert try-except.

```
# Debug the following code
def divide(a, b):
    return a / b

print(divide(10, 0))
```

Expected Output#3

- Corrected function with safe error handling

The screenshot shows a Google Colab notebook titled "AIPP_ASSIGNMENT-7.ipynb". In cell [4], the original code `print(divide(2, 0))` is run, resulting in a ZeroDivisionError. A tooltip explains that a try-except block can handle this error. In cell [5], the corrected code `safe_divide` is defined, which handles the error and prints a message. Test cases show the function working for non-zero divisions and failing for division by zero.

```
def safe_divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        print("Error: Cannot divide by zero!")
        return None # Or raise a custom error, or return a default value

# Test cases
print("2 divided by 0: ", safe_divide(2, 0))
print("10 divided by 2: ", safe_divide(10, 2))
print("7 divided by 3: ", safe_divide(7, 3))

Error: Cannot divide by zero!
2 divided by 0: None
10 divided by 2: 5.0
7 divided by 3: 2.3333333333333335
```

Q4.

Task Description#4

- Provide a faulty class definition (missing self in parameters). Let AI fix it

```
python

class Rectangle:
    def __init__(length, width):
        self.length = length
        self.width = width
```

Expected Output#4

- Correct `__init__()` method and explanation

The screenshot shows a Google Colab notebook titled "AIPP_ASSIGNMENT-7.ipynb". The code cell contains the following Python code:

```

class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

# Example of using the corrected class
my_rectangle = Rectangle(10, 5)
print(f"Rectangle length: {my_rectangle.length}, width: {my_rectangle.width}")

```

The output of the code is:

```

Rectangle length: 10, width: 5

```

The status bar at the bottom indicates it's running on Python 3.

Q5.

Task Description#5

- Access an invalid list index and use AI to resolve the Index Error.

python

```

numbers = [1, 2, 3]
print(numbers[5])

```

Expected Output#5

- AI suggests checking length or using safe access logic

The screenshot shows a Google Colab notebook titled "AIPP_ASSIGNMENT-7.ipynb". The code cell contains the following Python code:

```

numbers = [1, 2, 3]
invalid_index = 5
valid_index = 1

print("--- Using len() to check for valid index ---")
if invalid_index < len(numbers):
    print(f"Element at index {invalid_index}: {numbers[invalid_index]}")
else:
    print(f"Error: Index {invalid_index} is out of range for a list of length {len(numbers)}.")

if valid_index < len(numbers):
    print(f"Element at index {valid_index}: {numbers[valid_index]}")
else:
    print(f"Error: Index {valid_index} is out of range for a list of length {len(numbers)}.")

print("\n--- Using try-except to handle IndexError ---")
try:
    print(f"Attempting to access numbers[{invalid_index}]: {numbers[invalid_index]}")
except IndexError:
    print(f"Caught an IndexError: The index {invalid_index} is out of range!")

try:
    print(f"Attempting to access numbers[{valid_index}]: {numbers[valid_index]}")
except IndexError:
    print(f"Caught an IndexError: The index {valid_index} is out of range!")

print("\n--- Iterating safely ---")
for i, num in enumerate(numbers):
    print(f"Element at index {i}: {num}")

```

The status bar at the bottom indicates it's running on Python 3.

The screenshot shows a Jupyter Notebook interface running on Google Colab. The notebook is titled "AIPP_ASSIGNMENT-7.ipynb". The code cell contains the following Python code:

```
print("Caught an IndexError: The index [valid_index] is out of range!")
print("\n--- Iterating safely ---")
for i, num in enumerate(numbers):
    print(f"Element at index {i}: {num}")
...
--- Using len() to check for valid index ---
Error: Index 5 is out of range for a list of length 3.
Element at index 1: 2
...
--- Using try-except to handle IndexError ---
Caught an IndexError: The index 5 is out of range!
Attempting to access numbers[1]:
--- Iterating safely ---
Element at index 0: 1
Element at index 1: 2
Element at index 2: 3
```

The output pane shows the results of the code execution. The status bar at the bottom indicates the time is 5:58PM, the system temperature is 22°C, and the date is 09/11/2025.