

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: M.Tech. and MCA		Assignment Type: Lab	AcademicYear:2025-2026
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)			
<p style="text-align: center;">ASSIGNMENT – 7</p> <p style="text-align: center;">Name: Rimsha Mujeeb Roll Number:2503B05138 (M.Tech - CSE)</p>			
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 		

colab.research.google.com/drive/1a1Kaa879UpaRwUd086UCEp8IS0ewLa2M#scrollTo=48e1e46c

AIPP_ASSIGNMENT-7.ipynb

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all

AI-Error Debugging with AI: Systematic approaches to finding and fixing bugs

Task 1 prompt:

Study the code below, correct any syntax errors, take input values from the user, compute the sum as 'add', and print the value of 'add'.

```
def add(a,b)
return a+b
```

Enter the first number: 2
Enter the second number: 5
The sum is: 7.0

TASK 2 Prompt: Identify and fix a logic error in a loop that causes infinite iteration.

```
def count_down(n): while n >= 0: print(n) n += 1
```

Variables Terminal 5:58 PM Python 3

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

colab.research.google.com/drive/1a1Kaa879UpaRwUd086UCEp8IS0ewLa2M#scrollTo=48e1e46c

AIPP_ASSIGNMENT-7.ipynb

File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all

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 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.

```
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!")
```

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

Variables Terminal 5:58 PM Python 3

22°C Sunny 18:04 09/11/2023

Q3.

Task Description#3

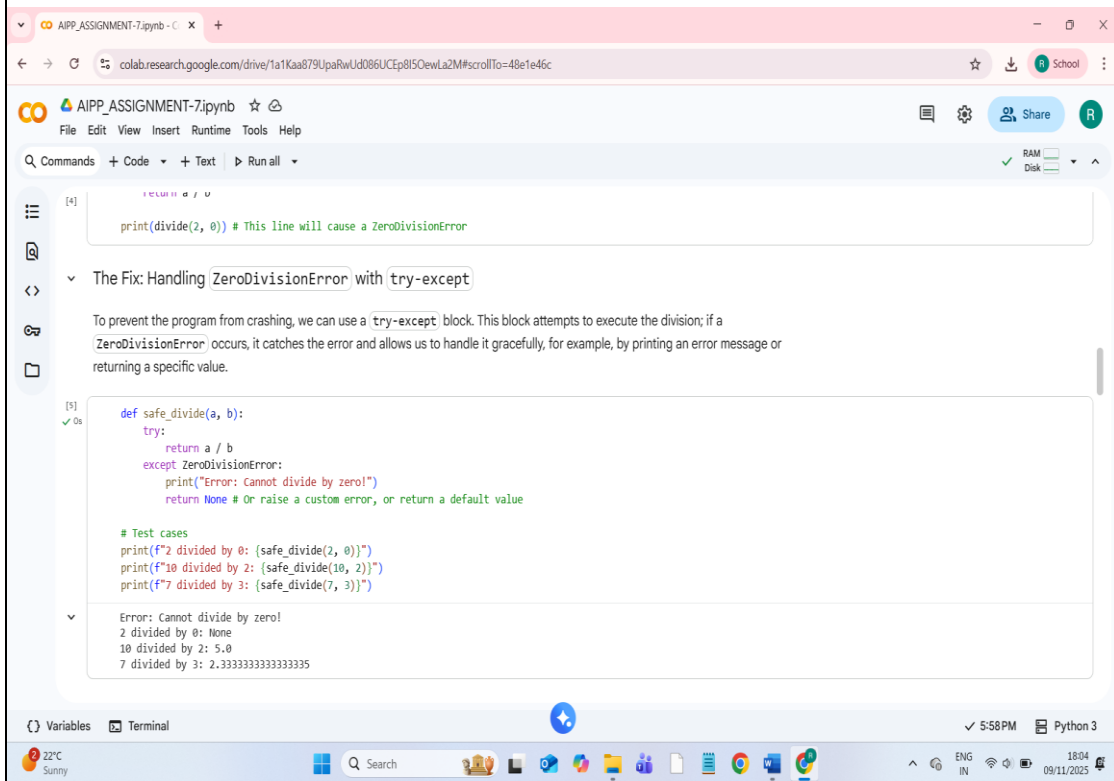
- Debug a runtime error caused by division by zero. Let AI insert try-except.

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

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

Expected Output#3

- Corrected function with safe error handling



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 Error and Fix for Rectangle class `__init__` method

The Error:

The original `__init__` method was defined as `def __init__(length, width):`. In Python, all instance methods, including the constructor `__init__`, must explicitly list `self` as their first parameter. This `self` parameter refers to the instance of the object being created, allowing you to assign attributes to it (e.g., `self.length = length`). Without `self`, the interpreter doesn't know to which object `length` and `width` should be assigned.

The Fix:

By adding `self` as the first parameter, `def __init__(self, length, width):`, we correctly define the method. Now, `self.length = length` and `self.width = width` correctly assign the provided `length` and `width` values as attributes to the new `Rectangle` object.

```
[6] ✓ 0s
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}")

Rectangle length: 10, width: 5
```

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

```
[6] ✓ 0s
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}")
```

AIPP_ASSIGNMENT-7.ipynb X

colab.research.google.com/drive/1a1Kaa879UpaRwUd086UCEp8I5OewLa2M#scrollTo=48e1e46c

AIPP_ASSIGNMENT-7.ipynb ☆

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

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}")
```

... --- 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[5]: 2

... --- Iterating safely ---
Element at index 0: 1
Element at index 1: 2
Element at index 2: 3

Variables Terminal

5:58 PM Python 3

22°C Sunny

Search

ENG IN 18:04 09/11/2023