LAB ASSIGNMENT-7.3

Hall Ticket no : *2303A51430*

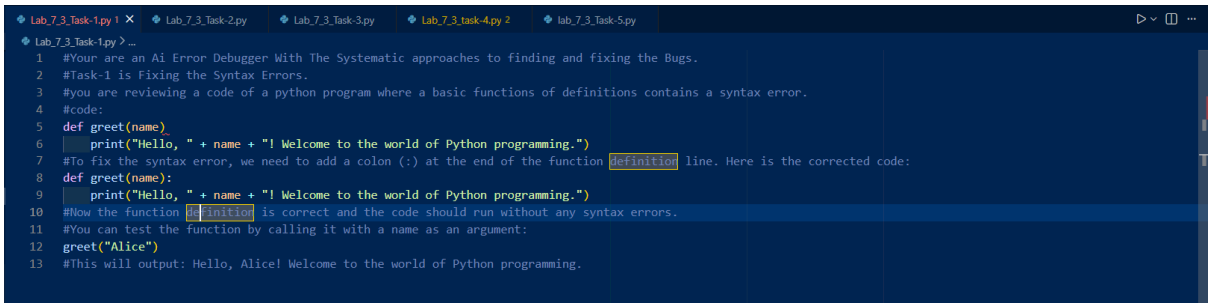
Name : *MADASI SATHWIK*

Task 1: Fixing Syntax Errors

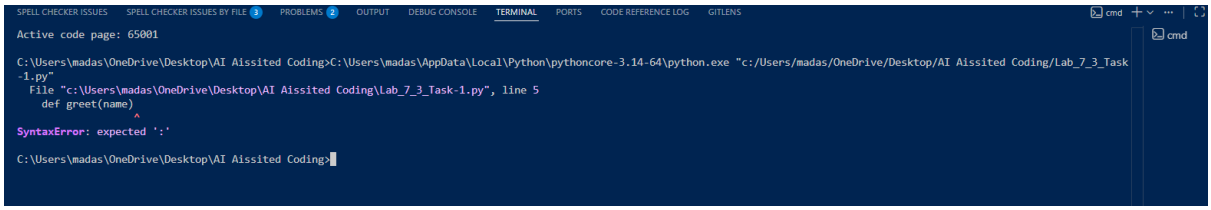
Scenario:

You are reviewing a Python program where a basic function definition contains a syntax error.

Prompt Used & code:



OUTPUT:



Explanation:

• The program defines a function named **greet** that is used to display a welcome message to a user.

• The function takes one parameter called **name**, which allows the message to be personalized.

• There was a syntax error because the function definition line was missing a colon at the end.

• In Python, a colon is mandatory after defining a function; otherwise, the interpreter throws a SyntaxError.

• After adding the colon, the function definition becomes correct and the program can run properly.

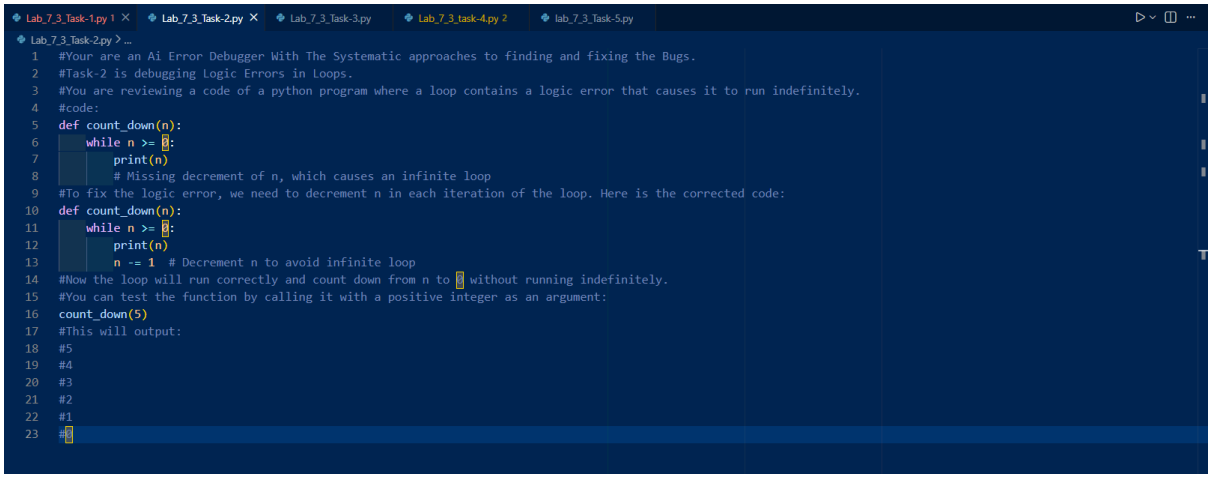
• When the function is called with the argument “Alice,” it prints the message:
Hello, Alice! Welcome to the world of Python programming.

Task 2 : Debugging Logic Errors in Loops

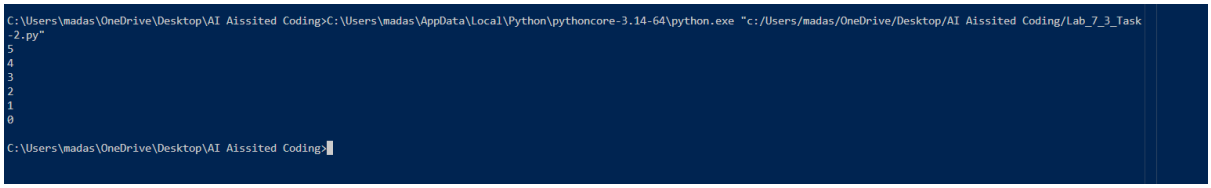
Scenario:

You are debugging a loop that runs infinitely due to a logical mistake.

Prompt & Code:



Output:



Explanation:

• The program defines a function named count_down that is designed to print numbers starting from a given value down to zero.

• The loop condition checks whether the value of n is greater than or equal to zero. As long as this condition is true, the loop continues running.

• In the original version, there was a logic error because the value of n was never reduced inside the loop.

• Since n was not decreasing, the condition always remained true, which caused the loop to run indefinitely (infinite loop).

• To fix this issue, the value of n is decreased by 1 during each iteration, ensuring that the loop eventually reaches zero and stops.

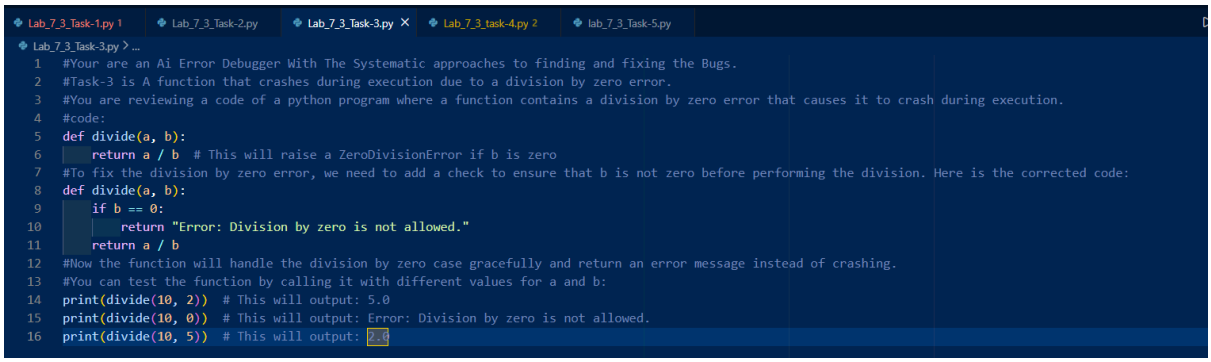
• **When the function is called with the value 5, it correctly prints the numbers 5, 4, 3, 2, 1, and 0, and then terminates properly without running forever.**

Task 3: Handling Runtime Errors (Division by Zero)

Scenario:

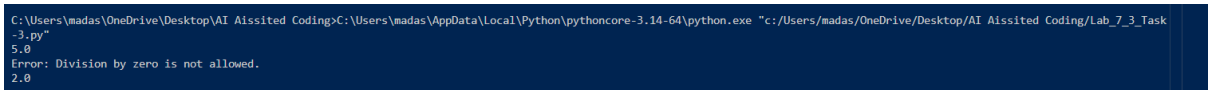
A Python function crashes during execution due to a division by zero error.

Prompt & Code:



```
#Your are an Ai Error Debugger With The Systematic approaches to finding and fixing the Bugs.
#Task-3 is A function that crashes during execution due to a division by zero error.
#You are reviewing a code of a python program where a function contains a division by zero error that causes it to crash during execution.
#code:
def divide(a, b):
    return a / b  # This will raise a ZeroDivisionError if b is zero
#To fix the division by zero error, we need to add a check to ensure that b is not zero before performing the division. Here is the corrected code:
def divide(a, b):
    if b == 0:
        return "Error: Division by zero is not allowed."
    return a / b
#Now the function will handle the division by zero case gracefully and return an error message instead of crashing.
#You can test the function by calling it with different values for a and b:
print(divide(10, 2))  # This will output: 5.0
print(divide(10, 0))  # This will output: Error: Division by zero is not allowed.
print(divide(10, 5))  # This will output: 2.0
```

Output:



```
C:\Users\madas\OneDrive\Desktop\AI Aissited Coding>C:\Users\madas\AppData\Local\Python\pythoncore-3.14-64\python.exe "c:/Users/madas/OneDrive/Desktop/AI Aissited Coding/Lab_7_3_Task
-3.py"
5.0
Error: Division by zero is not allowed.
2.0
```

Explanation:

• The program defines a function named **divide** that performs division between two numbers, **a** (numerator) and **b** (denominator).

• In the original version, the function directly divides **a by b** without checking whether **b** is zero.

• If **b is zero**, Python raises a **ZeroDivisionError**, which causes the program to crash during execution.

• To fix this runtime error, a condition is added to check whether **b equals zero** before performing the division.

• If **b is zero**, the function returns a clear error message instead of attempting the division, preventing the program from crashing.

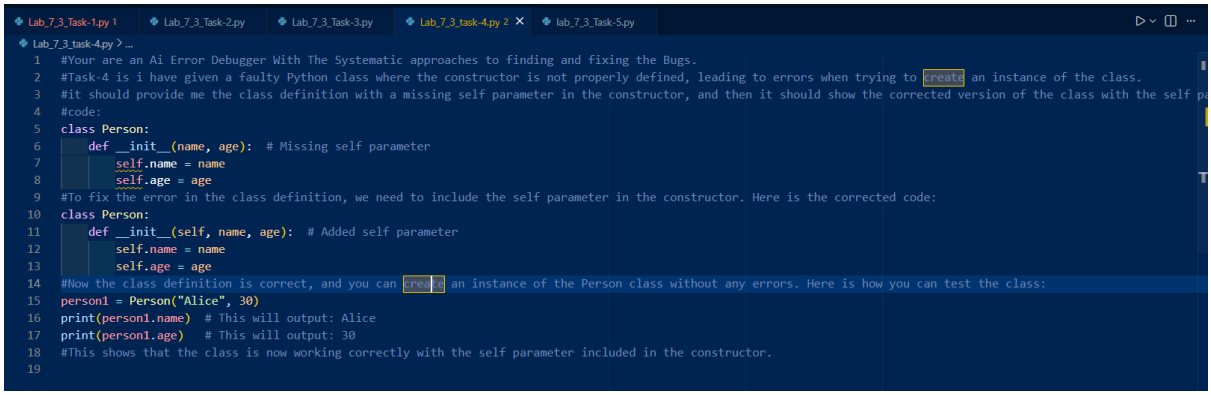
• When tested with different inputs, the function correctly returns the division result for valid values and safely handles the zero case by displaying an appropriate error message.

Task 4: Debugging Class Definition Errors

Scenario:

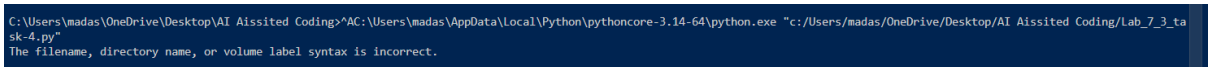
You are given a faulty Python class where the constructor is incorrectly defined.

Prompt & Code:



```
#Your are an Ai Error Debugger With The Systematic approaches to finding and fixing the Bugs.
#Task-4 is i have given a faulty Python class where the constructor is not properly defined, leading to errors when trying to create an instance of the class.
#it should provide me the class definition with a missing self parameter in the constructor, and then it should show the corrected version of the class with the self pa
#code:
class Person:
    def __init__(name, age):  # Missing self parameter
        self.name = name
        self.age = age
#To fix the error in the class definition, we need to include the self parameter in the constructor. Here is the corrected code:
class Person:
    def __init__(self, name, age):  # Added self parameter
        self.name = name
        self.age = age
#Now the class definition is correct, and you can create an instance of the Person class without any errors. Here is how you can test the class:
person1 = Person("Alice", 30)
print(person1.name)  # This will output: Alice
print(person1.age)   # This will output: 30
#This shows that the class is now working correctly with the self parameter included in the constructor.
```

Output:



```
C:\Users\madas\OneDrive\Desktop\AI Aissited Coding>^AC:\Users\madas\AppData\Local\Python\pythoncore-3.14-64\python.exe "c:/Users/madas/OneDrive/Desktop/AI Aissited Coding/Lab_7_3_ta
sk-4.py"
The filename, directory name, or volume label syntax is incorrect.
```

Explanation:

• The program defines a class named **Person**, which is used to represent a person with attributes like name and age.

• In the faulty version, the constructor was defined without including the **self** parameter, which is mandatory in Python class methods.

• The **self parameter** represents the current object (instance) of the class and allows access to its attributes and methods.

• Because **self was missing**, Python would raise an error when trying to create an object of the class, since the instance reference was not properly defined.

• To fix the issue, the **self parameter was added** as the first parameter in the constructor, allowing the object's data (name and age) to be stored correctly.

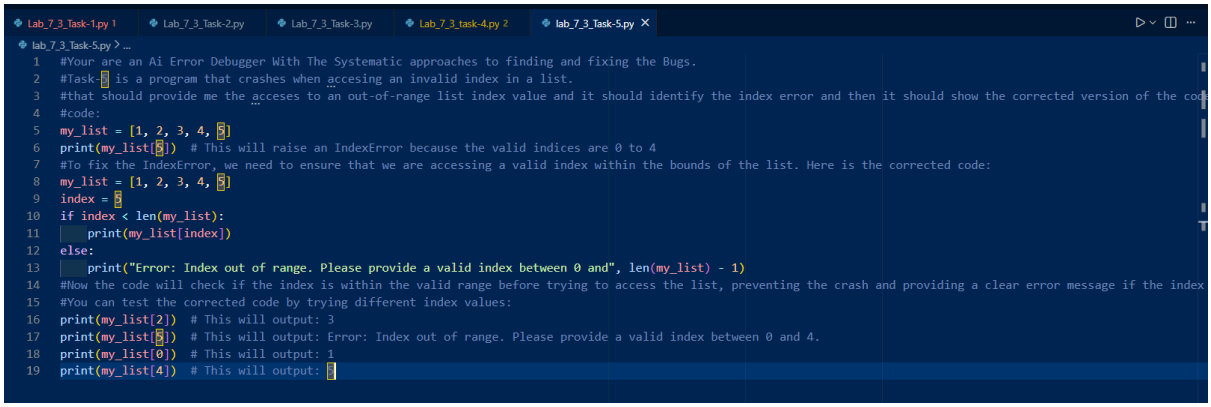
• After correcting the constructor, an instance of the Person class can be created successfully, and the stored attributes (name and age) can be accessed without any errors.

Task 5: Resolving Index Errors in Lists

Scenario:

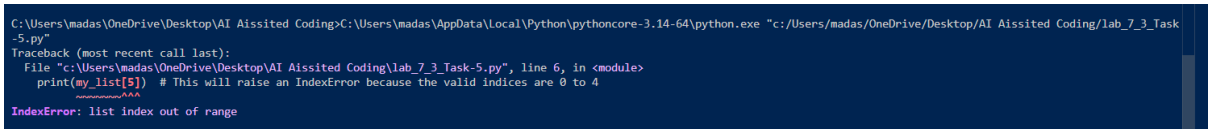
A program crashes when accessing an invalid index in a list.

Prompt & Code:



```
#Your are an Ai Error Debugger With The Systematic approaches to finding and fixing the Bugs.
#Task-5 is a program that crashes when accesing an invalid index in a list.
#that should provide me the acceses to an out-of-range list index value and it should identify the index error and then it should show the corrected version of the code
#code:
my_list = [1, 2, 3, 4, 5]
print(my_list[5])  # This will raise an IndexError because the valid indices are 0 to 4
#To fix the IndexError, we need to ensure that we are accessing a valid index within the bounds of the list. Here is the corrected code:
my_list = [1, 2, 3, 4, 5]
index = 5
if index < len(my_list):
    print(my_list[index])
else:
    print("Error: Index out of range. Please provide a valid index between 0 and", len(my_list) - 1)
#Now the code will check if the index is within the valid range before trying to access the list, preventing the crash and providing a clear error message if the index
#You can test the corrected code by trying different index values:
print(my_list[2])  # This will output: 3
print(my_list[5])  # This will output: Error: Index out of range. Please provide a valid index between 0 and 4.
print(my_list[0])  # This will output: 1
print(my_list[4])  # This will output: 5
```

Output:



```
C:\Users\madas\OneDrive\Desktop\AI Aissited Coding>C:\Users\madas\AppData\Local\Python\pythoncore-3.14-64\python.exe "c:/Users/madas/OneDrive/Desktop/AI Aissited Coding/lab_7_3_Task
-5.py"
Traceback (most recent call last):
  File "c:\Users\madas\OneDrive\Desktop\AI Aissited Coding\lab_7_3_Task-5.py", line 6, in <module>
    print(my_list[5])  # This will raise an IndexError because the valid indices are 0 to 4
          ~~~~~~~^^^
IndexError: list index out of range
```

Explanation:

• The program creates a list containing five elements, where the valid index positions range from 0 to 4.

• In the faulty version, the program tries to access the element at index 5, which does not exist in the list.

• Since index 5 is outside the valid range, Python raises an IndexError, causing the program to crash.

• To fix this issue, a condition is added to check whether the given index is less than the length of the list before accessing it.

• If the index is within the valid range, the program prints the corresponding element; otherwise, it displays a clear error message instead of crashing.

• After adding proper index validation, the program safely handles both valid and invalid index values, preventing runtime errors and improving reliability.