ASSIGNMENT-7.3

2303A51546

Batch-10

TASK-1:

Prompt:

I wrote a Python function, but it is showing a syntax error.
Please identify the syntax mistake, correct the function definition, and explain what caused the error.

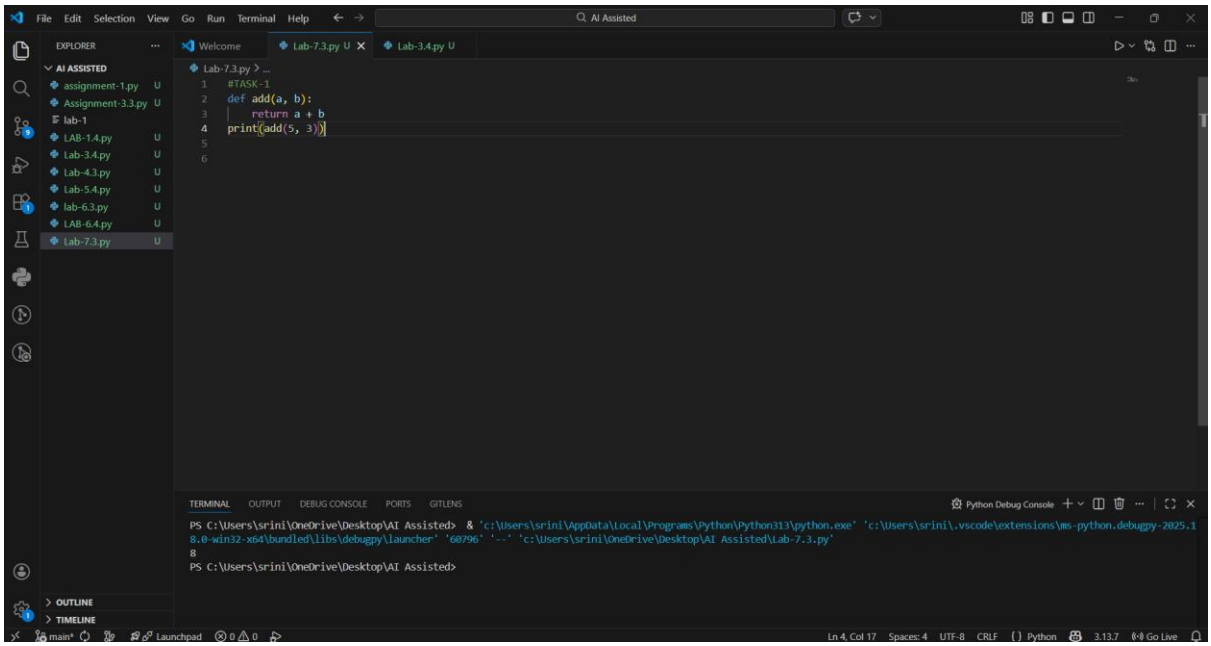
Wrong code with syntax error:

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

CODE:

```
def add(a, b):
  return a + b
print(add(5, 3))
```

Output:



Analysis:

In Python, every function definition must end with a colon (:).

The colon tells Python that the indented block below belongs to the function.

Without the colon, Python raises a SyntaxError and the program cannot run.

The AI correctly detected the missing colon and fixed the function.

TASK-2:

Prompt:

I wrote a Python function that is supposed to count down to 0, but it runs forever. Please find the logic error causing the infinite loop, correct the code, and explain the mistake.

Wrong code (Debugging logic errors in loops):

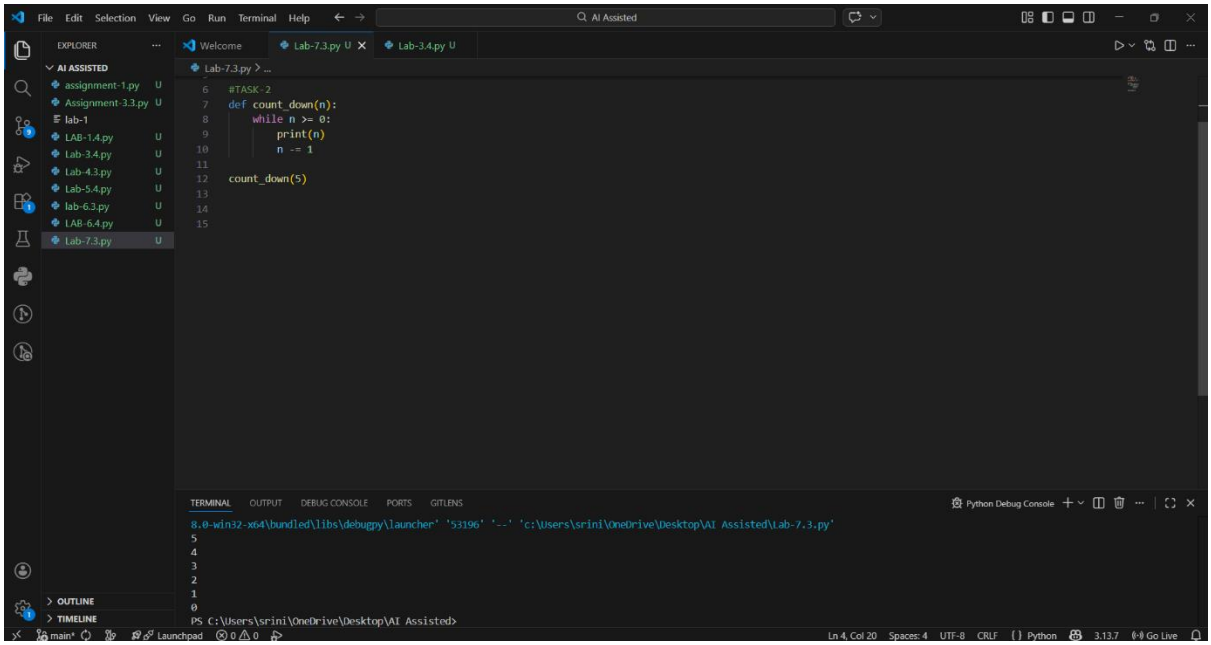
```
def count_down(n):
  while n >= 0:
    print(n)
    n += 1
```

CODE:

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


count_down(5)
```

Output:



Analysis:

The loop was intended to count down, but the variable was being incremented instead of decremented.

Because n kept increasing, it never became less than 0, so the loop never stopped.

Changing n += 1 to n -= 1 correctly decreases the value each time, allowing the loop to terminate.

TASK-3:

Prompt:

The following Python function crashes during execution.
Please identify the runtime error, fix the code using a try-except block, and explain how your solution prevents the crash.

Wrong code (handling runtime errors division by zero):

```
def divide(a, b):
  return a / b
print(divide(10, 0))
```

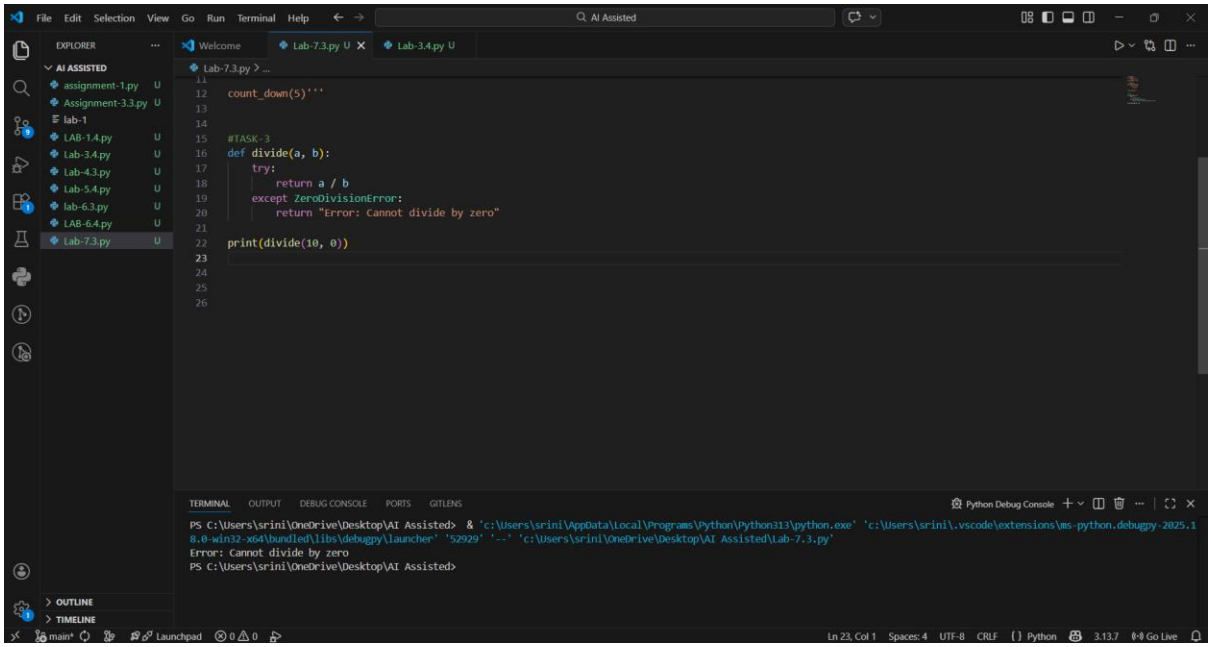
CODE:

```
def divide(a, b):
  try:
    return a / b
```

```
    except ZeroDivisionError:
        return "Error: Cannot divide by zero"


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

Output:



Analysis:

Division by zero is not allowed in Python and results in a runtime error.

A try block is used to run the risky code.

If a ZeroDivisionError occurs, the except block handles it safely.

Instead of crashing, the program now shows a user-friendly error message.

TASK-4:

Prompt:

I wrote this Python class, but it is not working correctly.
Please find the mistake in the constructor, fix the class definition, and explain why self is required.

Wrong code (Debugging Class Definition Errors):

```
class Rectangle:
    def __init__(length, width):
```

self.length = length

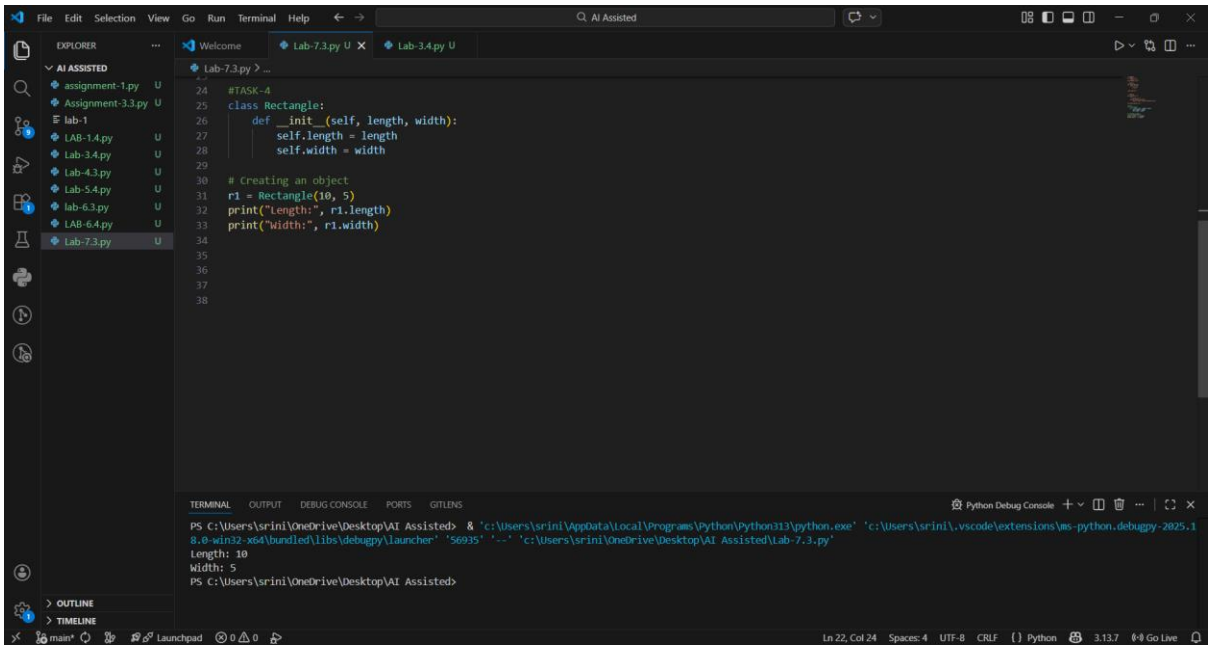
self.width = width

CODE:

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


# Creating an object
r1 = Rectangle(10, 5)
print("Length:", r1.length)
print("Width:", r1.width)
```

Output:



Analysis:

self represents the **current object** of the class.

When we create an object (r1 = Rectangle(10, 5)), Python automatically passes that object as self.

Using self.length and self.width stores the values inside the object.

Without self, the variables are not linked to the object, leading to errors.

TASK-5:

Prompt:

This Python program crashes with an index error when accessing a list element.
Please explain why the error occurs and modify the code to safely access the list using bounds checking or exception handling.

Wrong code (Resolving Index Errors in Lists):

numbers = [1, 2, 3]

print(numbers[5])

CODE:

numbers = [1, 2, 3]

index = 5

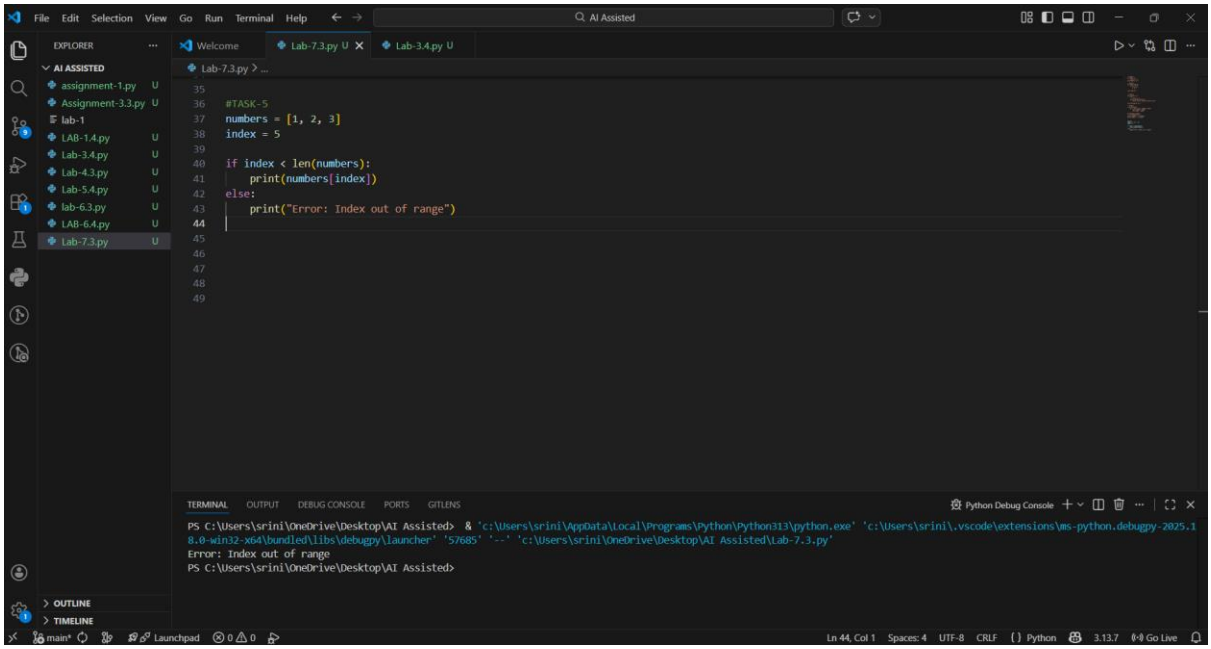
if index < len(numbers):

  print(numbers[index])

else:

  print("Error: Index out of range")

Output:



Analysis:

Python lists have a fixed range of valid positions based on their length.

Trying to access an element outside that range raises an IndexError.

We can prevent crashes by:

- Checking the index using len(list)
- Using a try-except block to catch the error safely