Name: Ch Pushkar

HT No: 2303A52362

Lab 7: Error Debugging with AI

Course: AI Assisted Coding (23CS002PC304)

Assignment: 7.3

Theme: Systematic approaches to finding and fixing bugs

Task 1: Fixing Syntax Errors

Description

The goal of this task is to identify and resolve a syntax error in a simple Python function. The provided code defines an addition function but is missing a specific syntactic element required by Python's grammar.

Prompt

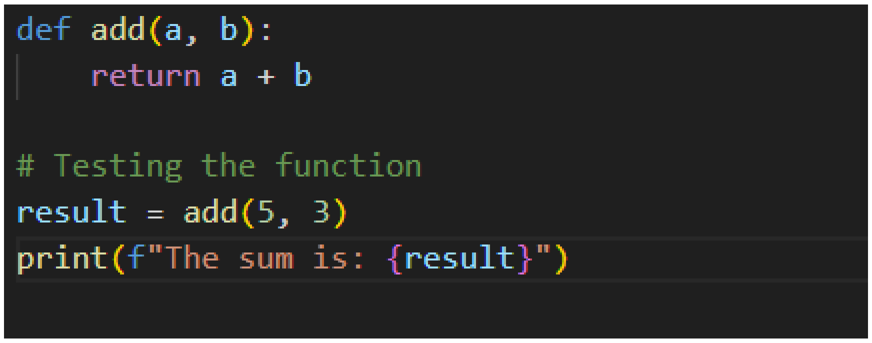
> "tried writing a basic function in Python that adds two numbers, but my code won't run—it throws a syntax error. Can you figure out what's missing and correct it "

Code

Faulty Code:

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

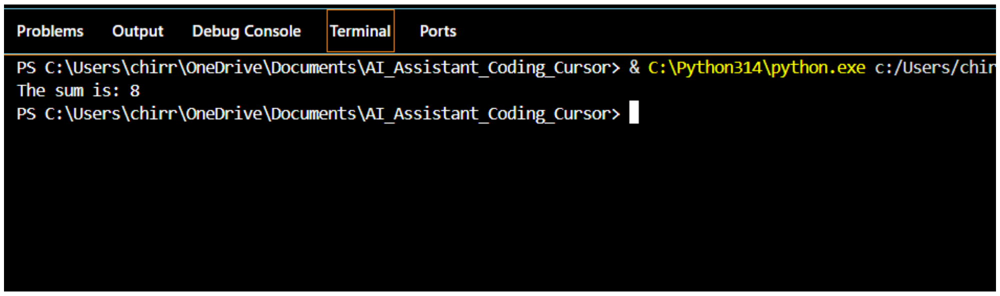
Corrected Code:



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

# Testing the function
result = add(5, 3)
print(f"The sum is: {result}")
```

Output



```
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor> & C:\Python314\python.exe c:/Users/chir
The sum is: 8
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor> 
```

Explanation

In Python, every function definition must end with a colon (`:`) to signal the start of its body. Without it, the interpreter doesn't recognize where the function's logic begins. Adding the colon is like telling Python, "Here's the setup—now comes the action.

Task 2: Debugging Logic Errors in Loops

Description

This task involves debugging a logic error in a while loop intended to count down. The original code creates an infinite loop because the counter is incremented instead of decremented, preventing the termination condition from ever being met.

Prompt

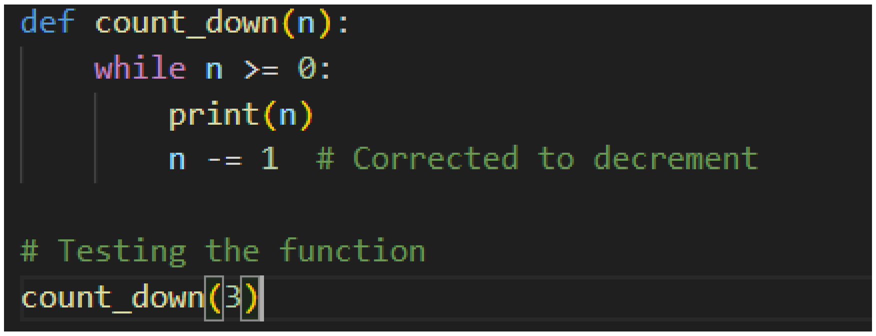
> My countdown loop never stops—it just keeps printing larger numbers instead of going down to zero. What's wrong with the loop logic, and how can I fix it?

Code

Faulty Code:

```
def
  count_down(n):
  while n >= 0:
    print(n)
    n += 1  # Logic error here
```

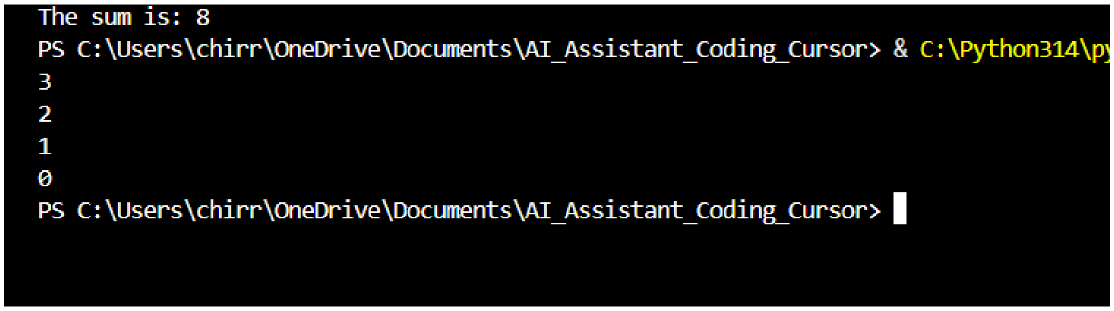
Corrected Code:



```python
def count_down(n):
    while n >= 0:
        print(n)
        n -= 1  # Corrected to decrement

# Testing the function
count_down(3)
```

Output



```
The sum is: 8
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor> & C:\Python314\py
3
2
1
0
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor>
```

Explanation

The loop was increasing `n` instead of decreasing it, so the condition `n >= 0` stayed true forever. It's like trying to empty a bucket by pouring more water into it. By changing `+=` to `-=`, we ensure the counter moves toward zero and eventually exits the loop.

Task 3: Handling Runtime Errors (Division by Zero)

Description

This task focuses on handling runtime errors that crash the program, specifically the "Division by Zero" error. The objective is to make the function robust using exception handling so the program can handle bad input gracefully.

Prompt

> "My division function works fine normally, but it crashes whenever someone tries to divide by zero. How can I make it handle this case without breaking the whole program?"

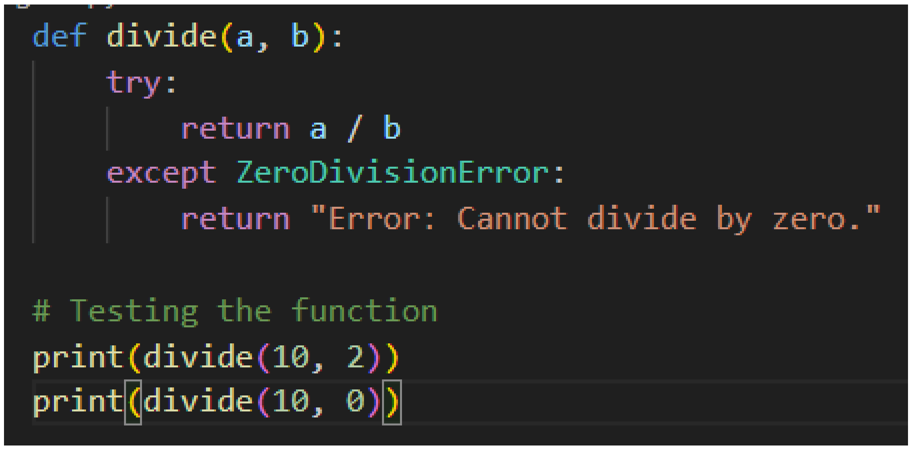
Code

Faulty Code:

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

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

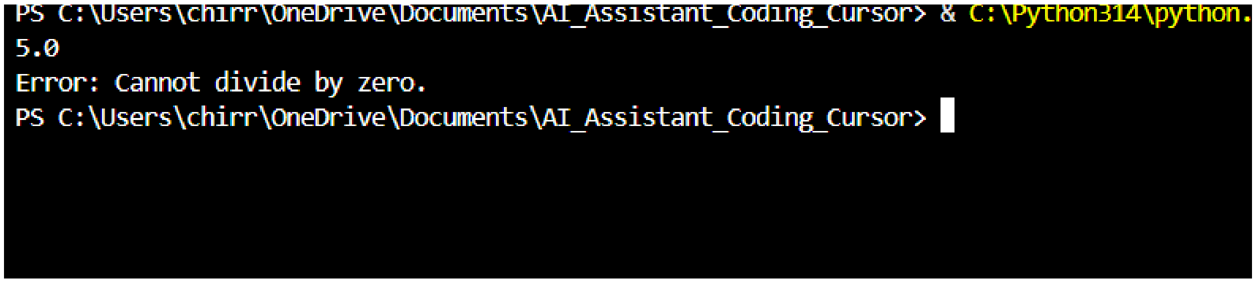
Corrected Code:



```
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."

# Testing the function
print(divide(10, 2))
print(divide(10, 0))
```

Output



```
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor> & C:\Python314\python.
5.0
Error: Cannot divide by zero.
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor>
```

Explanation

Division by zero is mathematically undefined and causes Python to raise a `ZeroDivisionError`. Using a `try-except` block acts like a safety guard—it attempts the operation, and if it detects a zero denominator, it returns a helpful message instead of crashing.

Task 4: Debugging Class Definition Errors

Description

This task involves fixing an Object-Oriented Programming (OOP) error. The provided class constructor is missing the self parameter, which is required in Python to refer to the specific instance of the object being created.

Prompt

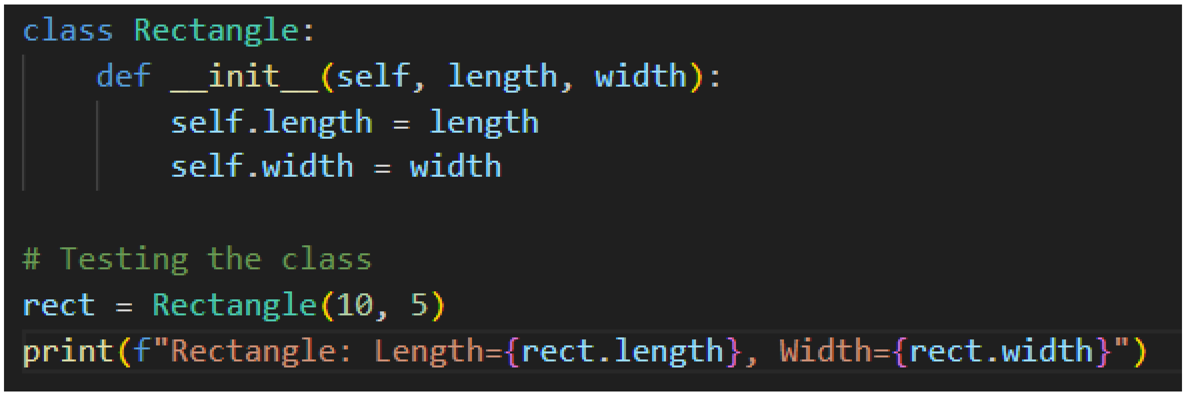
> "I created a `Rectangle` class, but when I try to set its dimensions, I get an error. It seems my `__init__` method isn't working right. Can you spot what's missing?"

Code

Faulty Code:

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

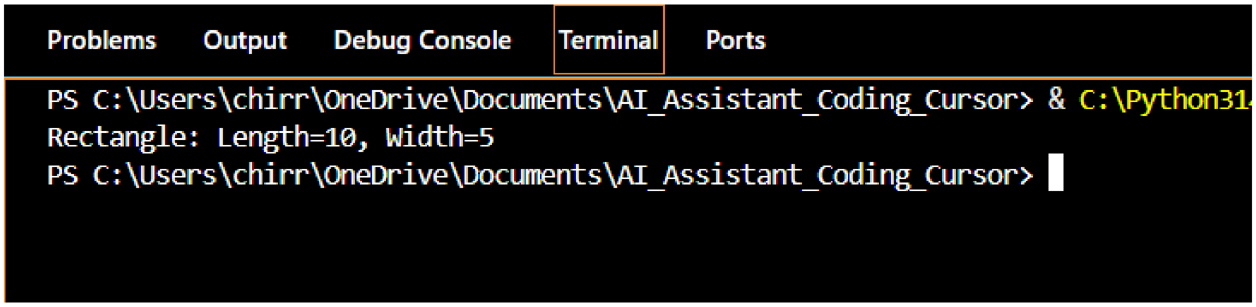
Corrected Code:



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

# Testing the class
rect = Rectangle(10, 5)
print(f"Rectangle: Length={rect.length}, Width={rect.width}")
```

Output



```
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor> & C:\Python31
Rectangle: Length=10, Width=5
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor> 
```

Explanation

In Python, every instance method—including `__init__`—must explicitly include `self` as its first parameter. This refers to the specific object being created. Without `self`, Python doesn't know which object's attributes you're trying to assign, leading to errors.

Task 5: Resolving Index Errors in Lists

Description

The final task addresses a common runtime error: IndexError. This occurs when trying to access a list element at a position that does not exist. The goal is to implement safe access methods using bounds checking.

Prompt

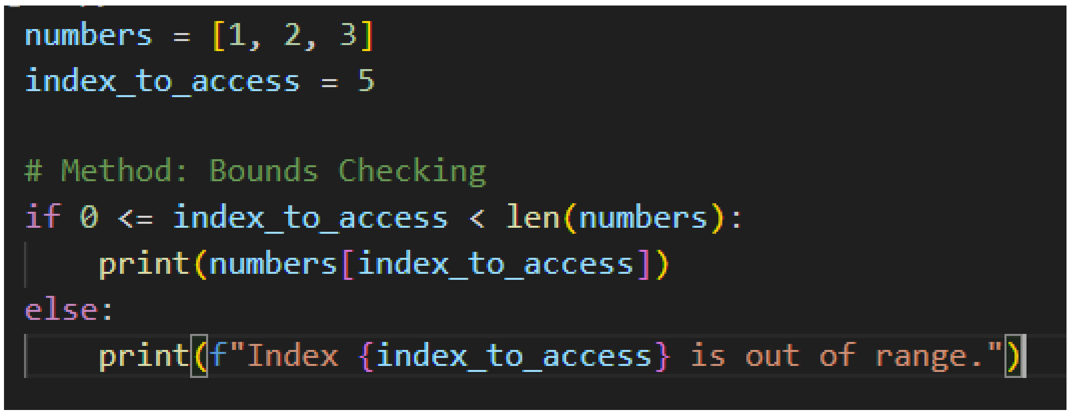
> "I have a list with 3 numbers, but my code crashes when I try to print an index that doesn't exist. Can you suggest a way to access list elements safely using an if-condition or exception handling?"

Code

Faulty Code:

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

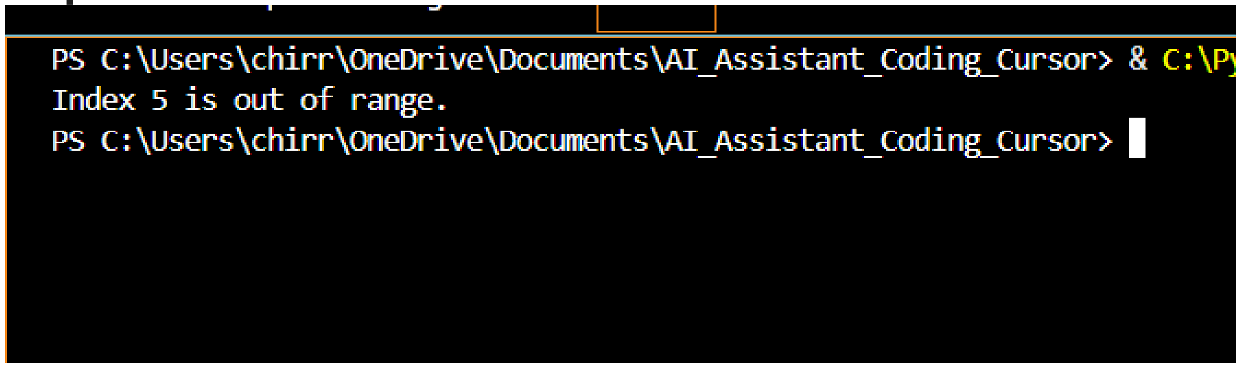
Corrected Code:



```python
numbers = [1, 2, 3]
index_to_access = 5

# Method: Bounds Checking
if 0 <= index_to_access < len(numbers):
    print(numbers[index_to_access])
else:
    print(f"Index {index_to_access} is out of range.")
```

Output



```
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor> & C:\Py
Index 5 is out of range.
PS C:\Users\chirr\OneDrive\Documents\AI_Assistant_Coding_Cursor>
```

Explanation

List indices in Python start at 0, so a 3-element list only has valid indices 0, 1, and 2. Accessing index 5 is like asking for the 6th item in a 3-item box—it doesn’t exist. By checking if the index is within the list’s length before accessing it, we prevent crashes and provide clear feedback.