ASSIGNMENT-07

NAME:K.Naga Chaitanya
HALL TICKET NO: 2303A51292
BATCH:05

Task 1: Fixing Syntax Errors

Scenario

def add(a,b) return
a+b

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

Requirements

- **Provide a Python function add(a, b) with a missing colon**
- **Use an AI tool to detect the syntax error**
- **Allow AI to correct the function definition**
- **Observe how AI explains the syntax issue**

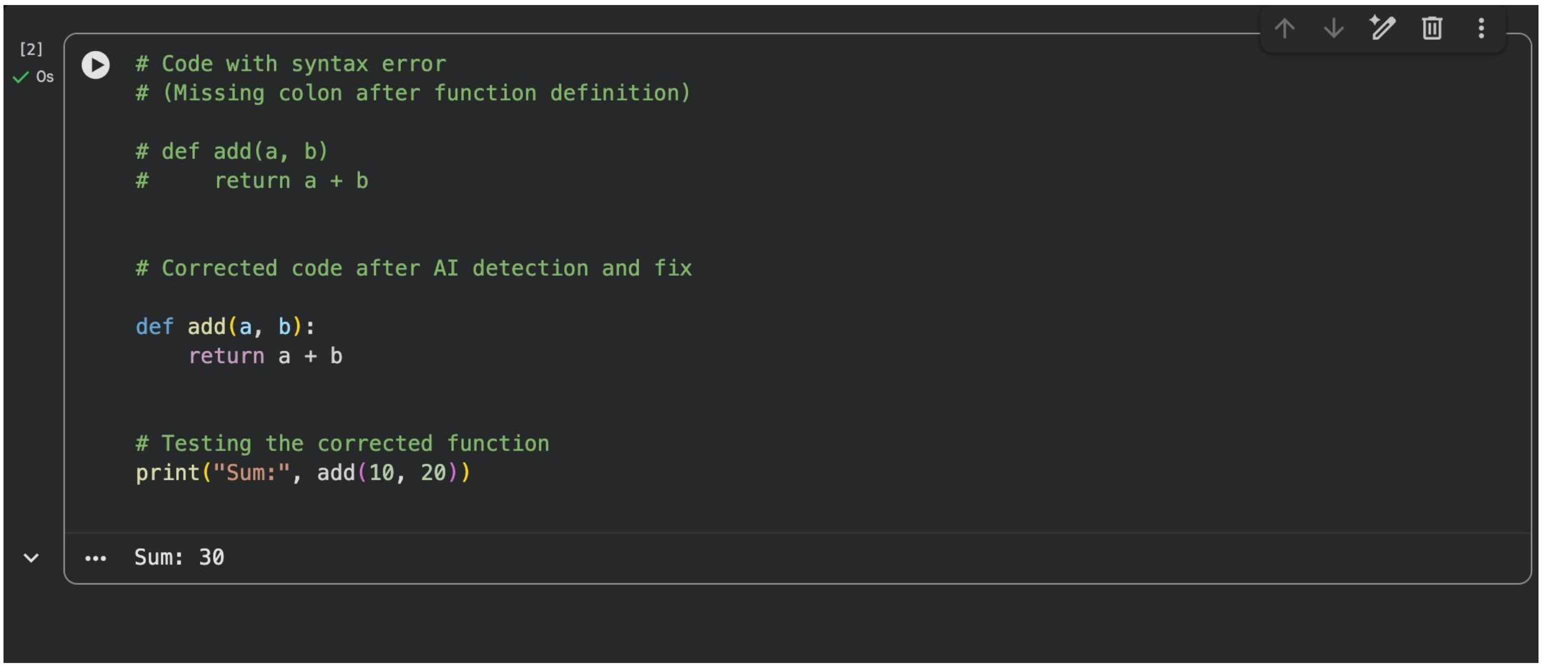
Expected Output

- **Corrected function with proper syntax**
- **Syntax error resolved successfully • AI-generated explanation of the fix**

PROMPT:

Identify and fix the syntax error in the following Python function.

CODE AND OUTPUT:



```python
# Code with syntax error
# (Missing colon after function definition)

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


# Corrected code after AI detection and fix

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


# Testing the corrected function
print("Sum:", add(10, 20))
```

```
Sum: 30
```

CODE EXPLANATION:
The syntax error occurs because the function definition is missing a colon (:) at the end of the def statement.
In Python, a colon is required after a function header to indicate the beginning of the function body.The AI tool detects this error, adds the missing colon, and corrects the function definition.After fixing the syntax, the function executes correctly and returns the sum of the two input values.

Task 2: Debugging Logic Errors in Loops

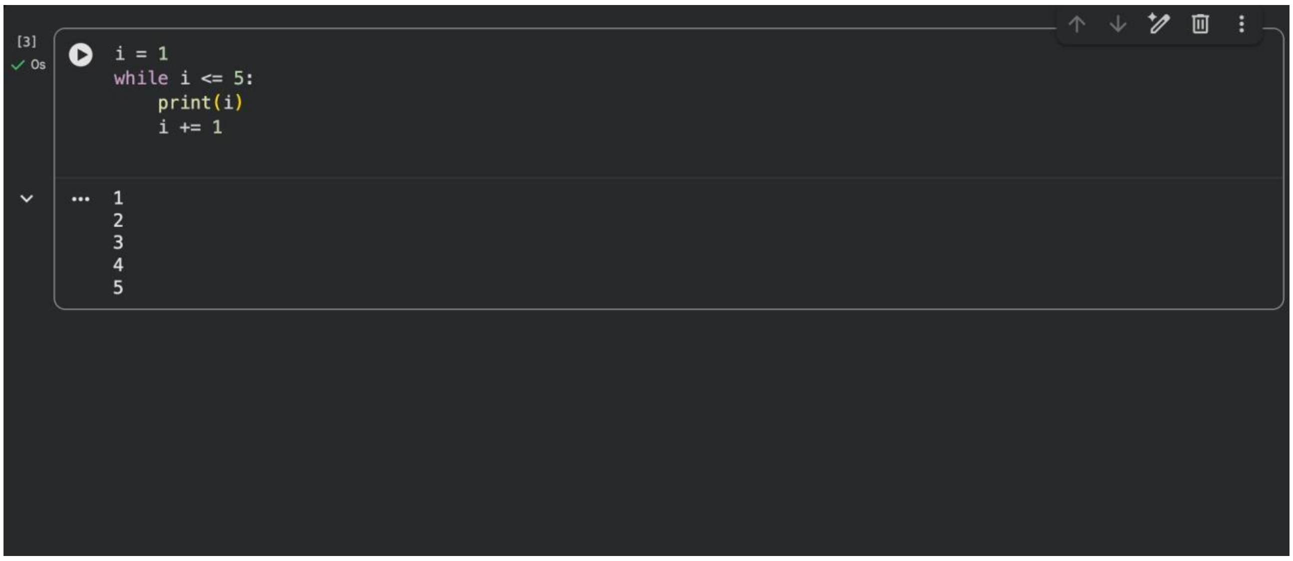
Scenario def
```
count_down(n):
    while n>=0:
        print(n)
        n+=1
```

You are debugging a loop that runs infinitely due to a logical mistake.
Requirements
- **Provide a loop with an increment or decrement error**
- **Use AI to identify the cause of infinite iteration**
- **Let AI fix the loop logic**
- **Analyze the corrected loop behavior**

Expected Output
- **Infinite loop issue resolved**
- **Correct increment/decrement logic applied**
- **AI explanation of the logic error**

PROMPT:
The following Python loop runs infinitely.
Identify the logic error, fix the loop, and explain why the infinite loop occurs.

CODE AND OUTPUT:



```
i = 1
while i <= 5:
    print(i)
    i += 1
```

```
1
2
3
4
5
```

CODE EXPLANATION:
The infinite loop occurs because the loop control variable i is never updated inside the loop.
Since i remains equal to 1, the condition i <= 5 is always true, causing the loop to run indefinitely.
The AI tool identifies this logic error and fixes it by adding an increment statement i += 1.
After correction, the value of i increases in each iteration, and the loop terminates once i becomes greater than 5.

Task 3: Handling Runtime Errors (Division by Zero)

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

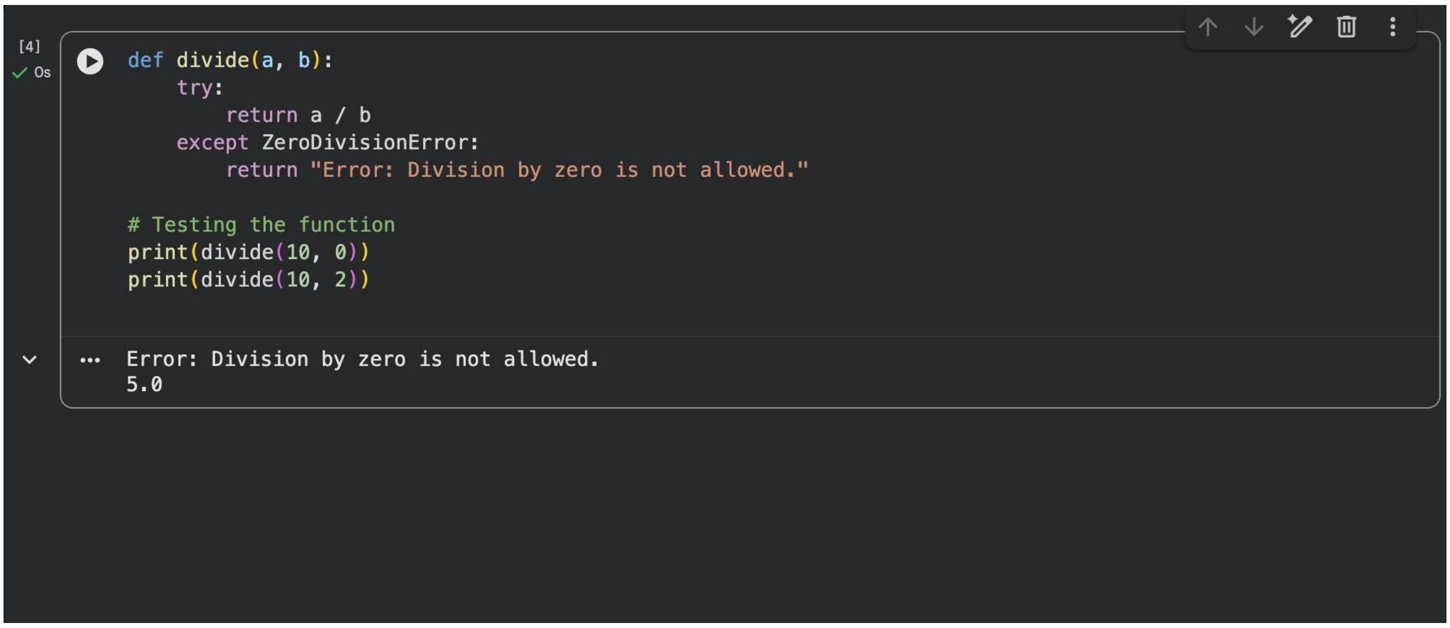
A Python function crashes during execution due to a division by zero error.
Requirements
- **Provide a function that performs division without validation**
- **Use AI to identify the runtime error**
- **Let AI add try-except blocks for safe execution**
- **Review AI's error-handling approach**

Expected Output
- **Function executes safely without crashing**
- **Division by zero handled using try-except**
- **Clear AI-generated explanation of runtime error handling**

PROMPT:
Identify the error and modify the code to handle it safely using try-except.

☐

CODE AND OUTPUT:



```
[4]
def divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Division by zero is not allowed."

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

```
Error: Division by zero is not allowed.
5.0
```

CODE EXPLANATION:
The function `divide(a, b)` performs division of two numbers and may cause a runtime error when the divisor is zero. Python raises a `ZeroDivisionError` in such cases, which can crash the program. To handle this safely, a `try-except` block is used. The `try` block executes the division, while the `except` block catches the error and returns a meaningful message. This approach ensures safe execution and prevents program termination.

Task 4: Debugging Class Definition Errors
Scenario

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

You are given a faulty Python class where the constructor is incorrectly defined.
Requirements
• Provide a class definition with missing self-parameter
• Use AI to identify the issue in the __init__() method

- **Allow AI to correct the class definition**
- **Understand why self is required**

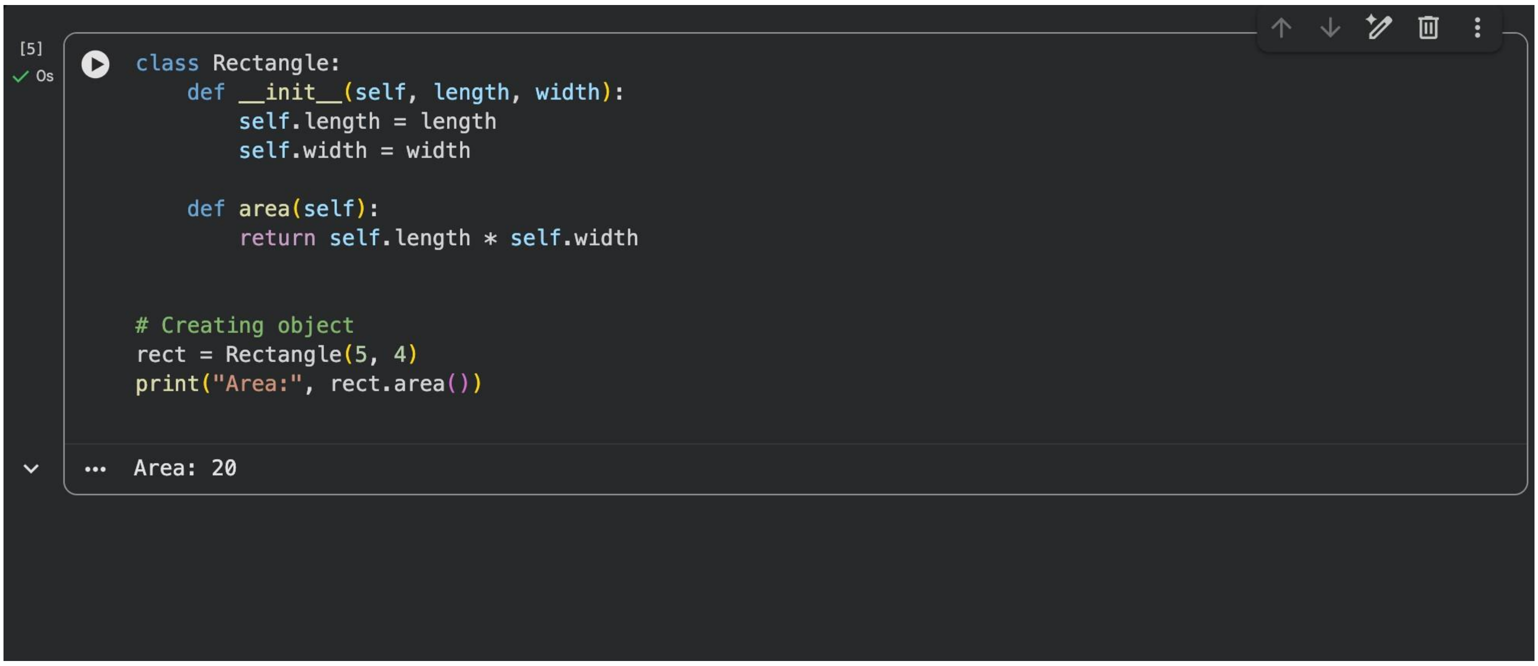
Expected Output

- **Corrected __init__() method**
- **Proper use of self in class definition • AI explanation of object-oriented error**

PROMPT:

The following Python class has an error in the constructor.
Identify the problem, fix the class definition, and explain why the error occurs.

CODE AND OUTPUT:



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

    def area(self):
        return self.length * self.width


# Creating object
rect = Rectangle(5, 4)
print("Area:", rect.area())
```

```
Area: 20
```

CODE EXPLANATION:

The error in the `Rectangle` class occurs because the constructor `__init__()` is defined without the `self` parameter. In Python, `self`is required to refer to the current object and to store data inside the object. Without using `self`, the variables `length` and `width`remain local to the constructor and are not saved as object attributes. The AI identifies this issue and fixes it by adding `self`and assigning the values to `self.length` and `self.width`. After correction, each `Rectangle` object correctly stores its dimensions and can compute the area without errors.

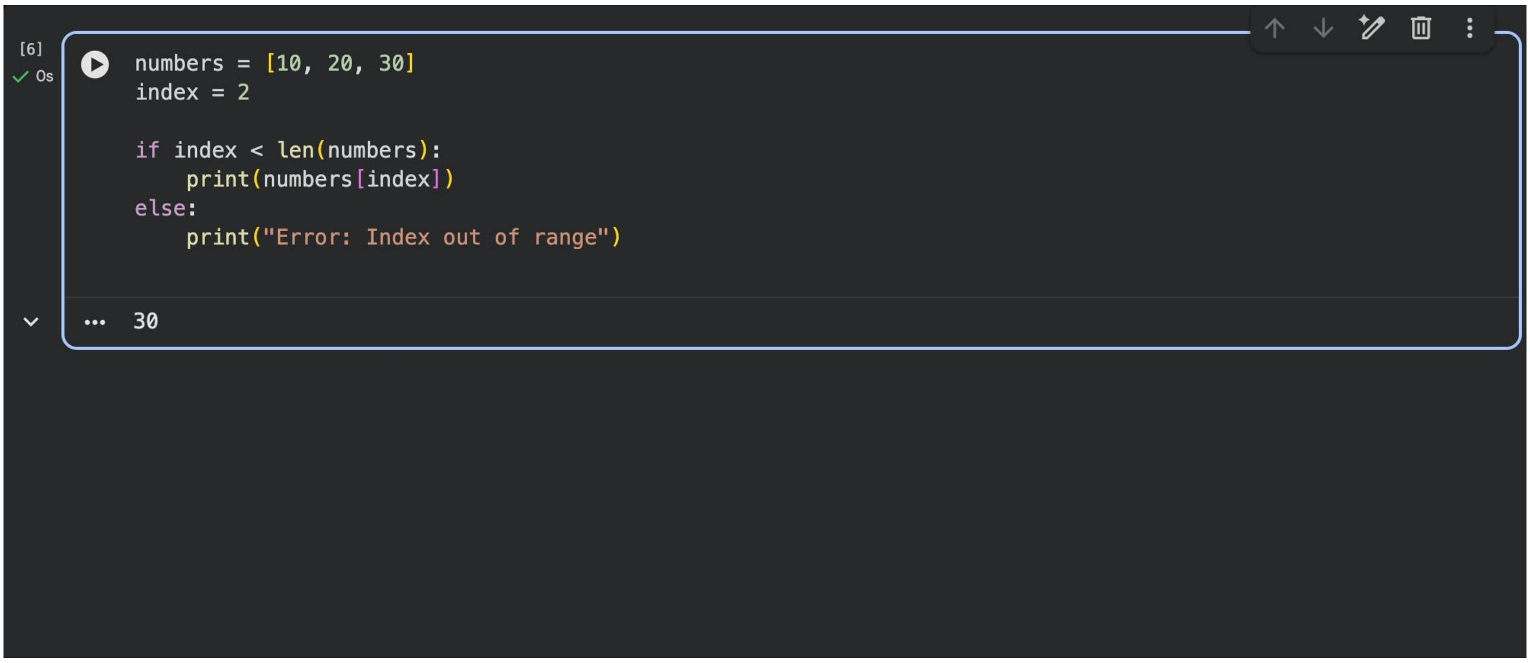
Task 5: Resolving Index Errors in Lists

Scenario numbers =
[10, 20, 30]
print(numbers[5])

A program crashes when accessing an invalid index in a list.
Requirements
- **Provide code that accesses an out-of-range list index**
- **Use AI to identify the Index Error**
- **Let AI suggest safe access methods**
- **Apply bounds checking or exception handling**

Expected Output
- **Index error resolved**
- **Safe list access logic implemented**
- **AI suggestion using length checks or exception handling**

□
PROMPT:
The following Python program crashes with an IndexError. Identify the error, suggest a safe way to access the list, and fix the code using bounds checking or exception handling. **CODE AND OUTPUT:**



```
[6]
✓ 0s
numbers = [10, 20, 30]
index = 2

if index < len(numbers):
    print(numbers[index])
else:
    print("Error: Index out of range")
```

```
30
```

CODE EXPLANATION:
The program crashes because it tries to access an index that does not exist in the list. Python lists are zero-indexed, so a list with three elements has valid indices from 0 to 2. Accessing index 5 raises an `IndexError`. The AI identifies this issue and suggests safe access methods such as checking the

index against the list length or using a `try-except` block. These approaches prevent the program from crashing and ensure safe list access.