# ASSIGNMENT – 7.3

B.Varun

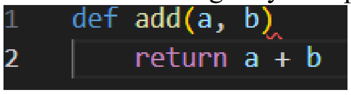
2303A52020

Batch – 38

**Task 1: Fixing Syntax Errors**

**Scenario**

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



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

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

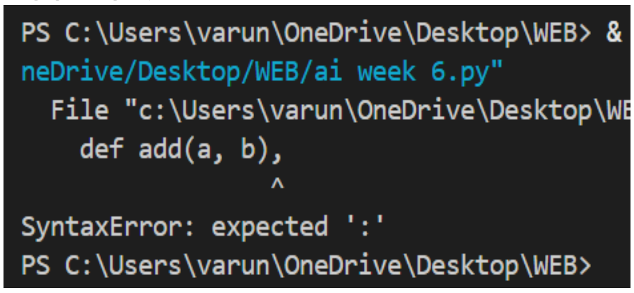
**Expected Output**

• Corrected function with proper syntax

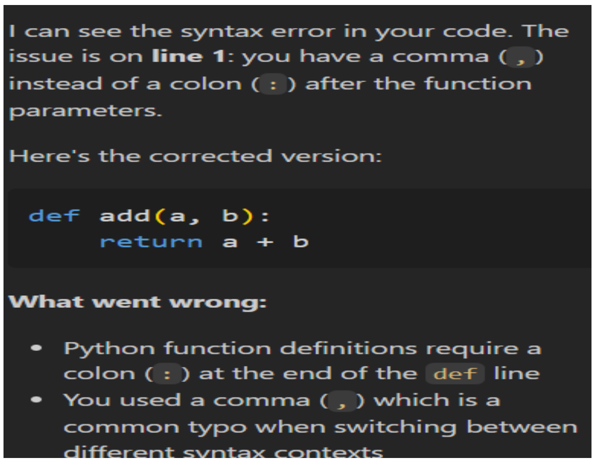
• Syntax error resolved successfully

• AI-generated explanation of the fix

**OUTPUT:**



```
PS C:\Users\varun\OneDrive\Desktop\WEB> &
neDrive/Desktop/WEB/ai week 6.py"
  File "c:\Users\varun\OneDrive\Desktop\WE
    def add(a, b),
                 ^
SyntaxError: expected ':'
PS C:\Users\varun\OneDrive\Desktop\WEB>
```



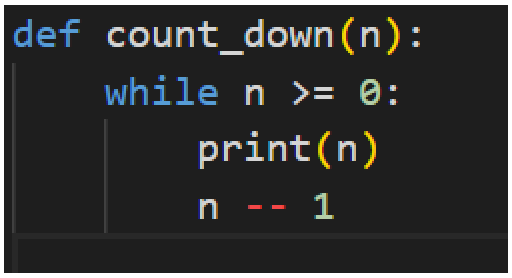
I can see the syntax error in your code. The issue is on **line 1**: you have a comma ( `,` ) instead of a colon ( `:` ) after the function parameters.

Here's the corrected version:

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

**What went wrong:**

- Python function definitions require a colon ( `:` ) at the end of the `def` line
- You used a comma ( `,` ) which is a common typo when switching between different syntax contexts



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

**Task 2: Debugging Logic Errors in Loops**

**Scenario**

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

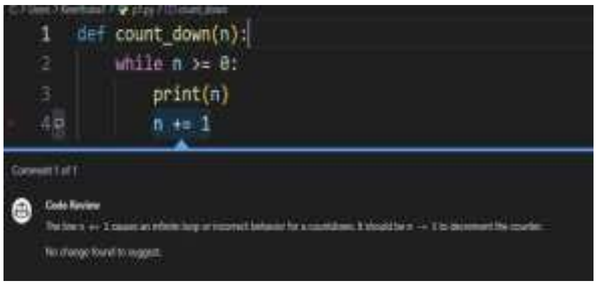
Requirements

- 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

**OUTPUT:**



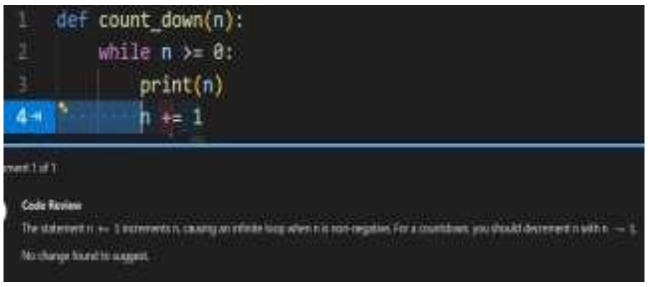
```
def count_down(n):
    while n >= 0:
        print(n)
        n += 1
```

Comment 1 of 1

Code Review

The line n += 1 causes an infinite loop or incorrect behavior for a countdown. It should be n -= 1 to decrement the counter.

No change found to suggest.



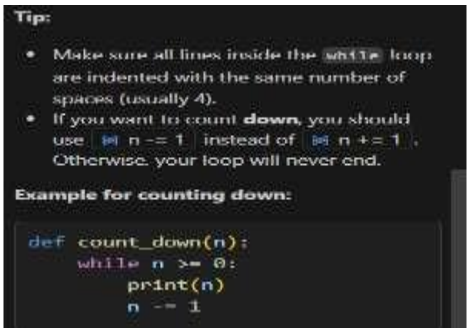
```
def count_down(n):
    while n >= 0:
        print(n)
        n += 1
```

Comment 1 of 1

Code Review

The statement n += 1 increments n, causing an infinite loop when n is non-negative. For a countdown, you should decrement n with n -= 1.

No change found to suggest.



**Tip:**

- Make sure all lines inside the `while` loop are indented with the same number of spaces (usually 4).
- If you want to count **down**, you should use `n -= 1` instead of `n += 1`. Otherwise, your loop will never end.

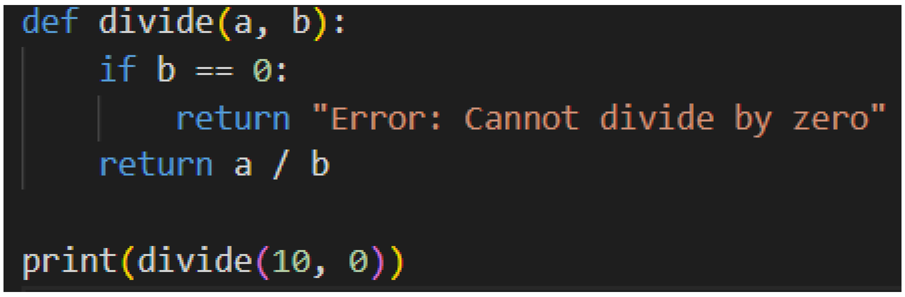
**Example for counting down:**

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

**Task 3: Handling Runtime Errors (Division by Zero)**

**Scenario**

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



```
def divide(a, b):
    if b == 0:
        return "Error: Cannot divide by zero"
    return a / b

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

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

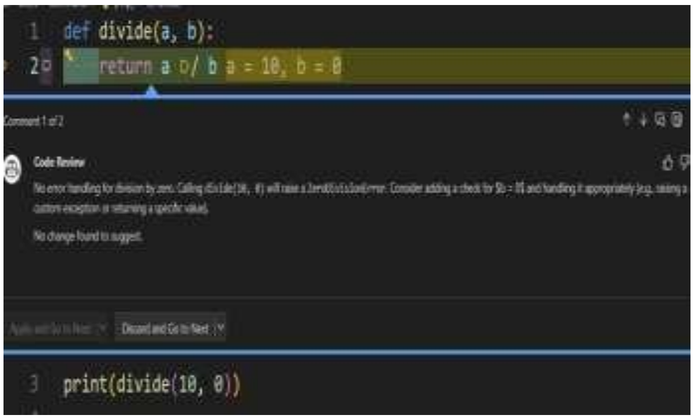
• Review AI's error-handling approach

**Expected Output**

• Function executes safely without crashing

• Division by zero handled using try-except

• Clear AI-generated explanation of runtime error handling



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

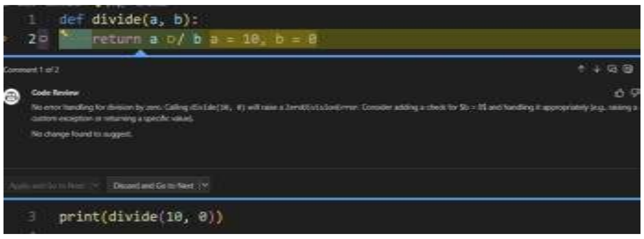
Comment 1 of 2

Code Review

No error handling for division by zero. Calling divide(10, 0) will raise a ZeroDivisionError. Consider adding a check for `b == 0` and handling it appropriately (e.g., raising a custom exception or returning a specific value).

No change found to suggest.

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



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

Comment 1 of 2

Code Review

No error handling for division by zero. Calling divide(10, 0) will raise a ZeroDivisionError. Consider adding a check for `b == 0` and handling it appropriately (e.g., raising a custom exception or returning a specific value).

No change found to suggest.

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

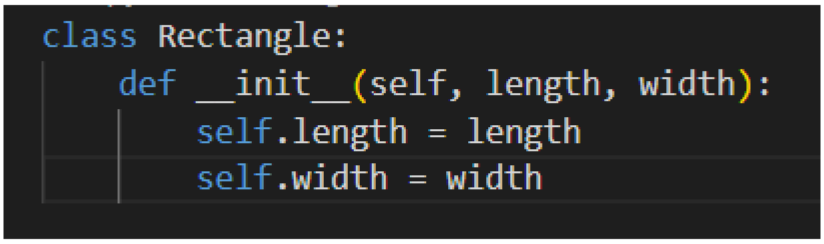
**Explanation:**
The runtime error occurs when dividing by zero.
Using a try-except block catches the ZeroDivisionError and prevents the program from crashing.

### Task 4: Debugging Class Definition Errors

Scenario

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

Requirements



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

• 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

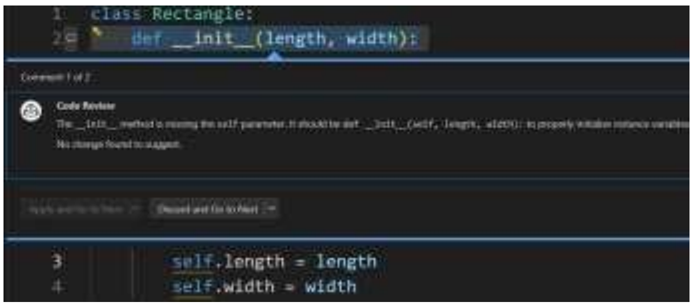
• Understand why self is required

**Expected Output**

• Corrected __init__() method

• Proper use of self in class definition

• AI explanation of object-oriented error



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

Comment 1 of 2

Code Review

The __init__ method is missing the self parameter. It should be def __init__(self, length, width): to properly initialize instance variables

No change found to suggest.

```
        self.length = length
        self.width = width
```

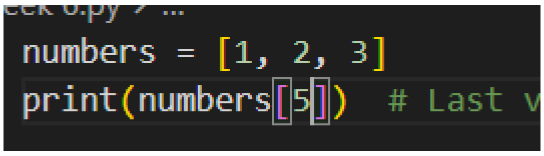
**EXPLANATION:**
The constructor __init__was missing the self parameter.
self refers to the current object and is required to store and access instance variables inside a class.

**Task 5: Resolving Index Errors in Lists**

Scenario

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

Requirements



```
eek 6.py > ...
numbers = [1, 2, 3]
print(numbers[5])  # Last v
```

• Provide code that accesses an out-of-range list index

• Use AI to identify the Index Error

• Let AI suggest safe access methods

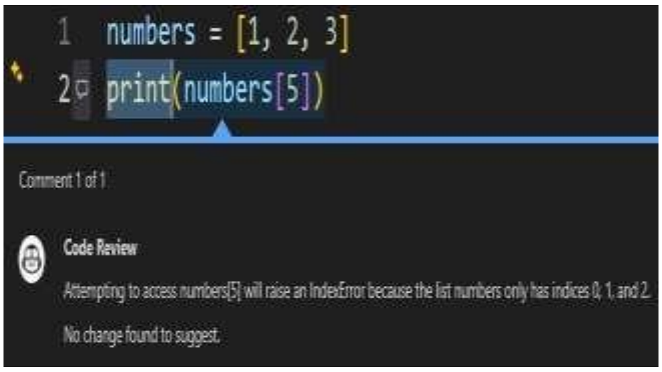
• Apply bounds checking or exception handling

**Expected Output**

• Index error resolved

• Safe list access logic implemented

• AI suggestion using length checks or exception handling



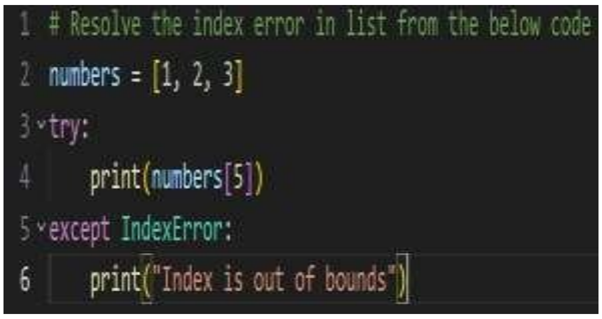
```
numbers = [1, 2, 3]
print(numbers[5])
```

Comment 1 of 1

Code Review

Attempting to access numbers[5] will raise an IndexError because the list numbers only has indices 0, 1, and 2.

No change found to suggest.



```
# Resolve the index error in list from the below code
numbers = [1, 2, 3]
try:
    print(numbers[5])
except IndexError:
    print("Index is out of bounds")
```

**Explanation:**
The error occurs because index 5 does not exist in the list.
Using try-except catches the Index Error and prevents the program from crashing.