

AI Assistant Coding

Assignment-7.3

Name: Aman Sarkar

Ht.no: 2303A51273

Batch: 05

Task 1: Fixing Syntax Errors



The screenshot shows a code editor with a Python function definition that has a syntax error. The code is:

```
[1] -def add(a,b)
+def add(a,b):
    return a+b
```

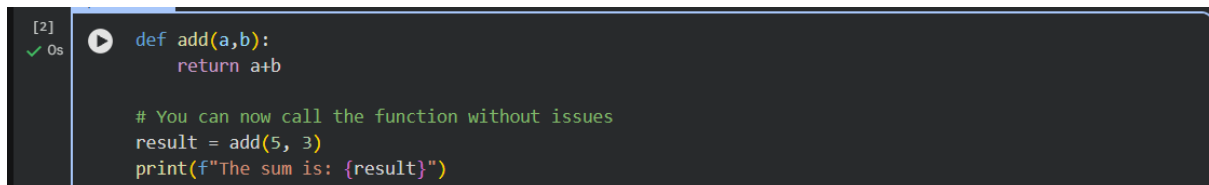
The error message is:

```
... File "/tmp/ipython-input-193546707.py", line 1
      def add(a,b)
            ^
SyntaxError: expected ':'
```

Next steps: [Explain error](#)

The code above has a `SyntaxError`. When you run it, Python will raise an error because the `def` statement (function definition) is missing a colon `:` at the end. Every function definition in Python must end with a colon before the indented function body.

Code:

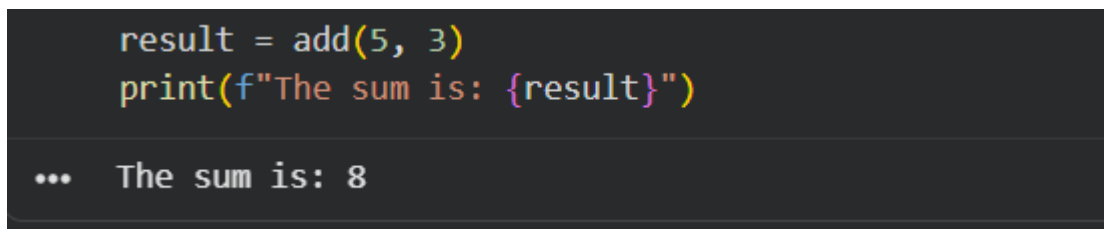


The screenshot shows the corrected code in a code editor:

```
[2] def add(a,b):
    return a+b

# You can now call the function without issues
result = add(5, 3)
print(f"The sum is: {result}")
```

Output:



The screenshot shows the output of the corrected code in a code editor:

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

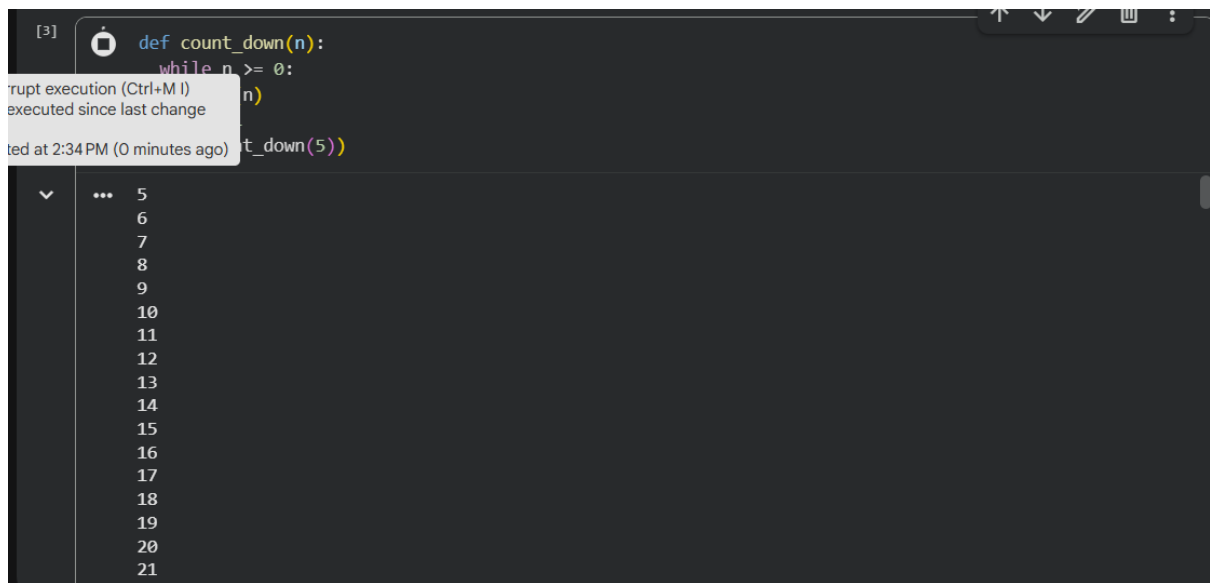
... The sum is: 8
```

Code Explanation:

Colon (`:`) is mandatory at the end of function definitions, loops, and conditionals in Python.

- It signals the start of a block of code that must be indented.
- Forgetting it leads to a `SyntaxError`.

Task 2: Debugging Logic Errors in Loops

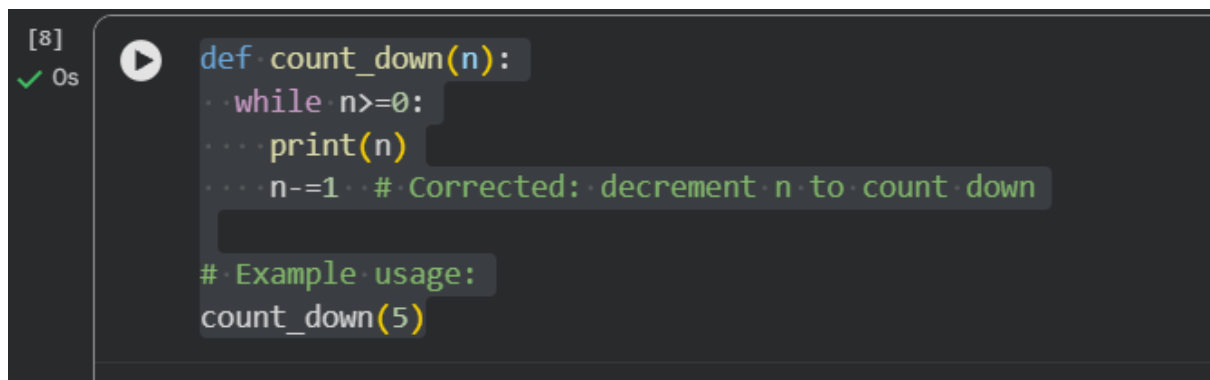


The screenshot shows a Jupyter Notebook cell with a Python function `count_down(n)` that has a logic error. The function is defined as follows:

```
def count_down(n):  
    while n >= 0:  
        print(n)  
        n  
count_down(5)
```

A tooltip indicates that the execution was interrupted (Ctrl+M) and that the code was not executed since the last change. The output of the cell shows a list of numbers from 5 to 21, which is incorrect for the intended function.

Code:

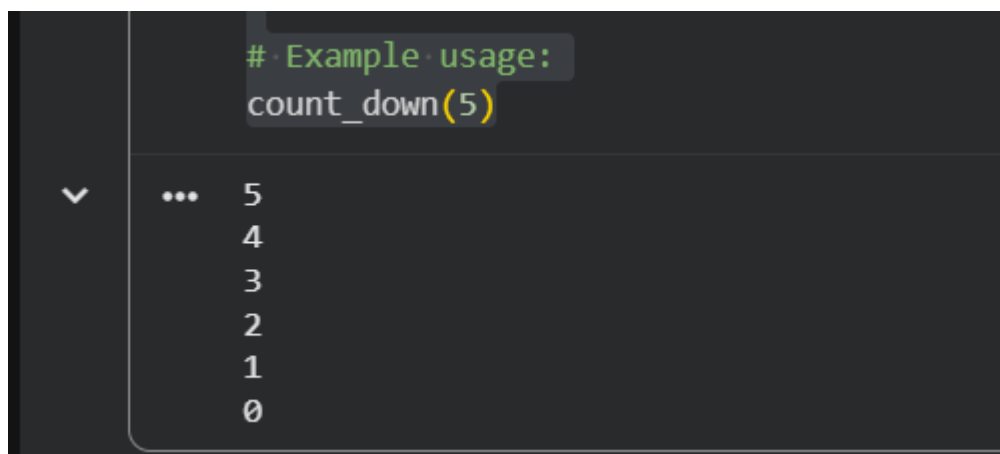


The screenshot shows a Jupyter Notebook cell with the corrected `count_down(n)` function. The function is defined as follows:

```
def count_down(n):  
    while n >= 0:  
        print(n)  
        n -= 1  # Corrected: decrement n to count down  
  
# Example usage:  
count_down(5)
```

The cell is marked as executed successfully (green checkmark) and the output is empty, indicating that the function is working correctly.

Output:



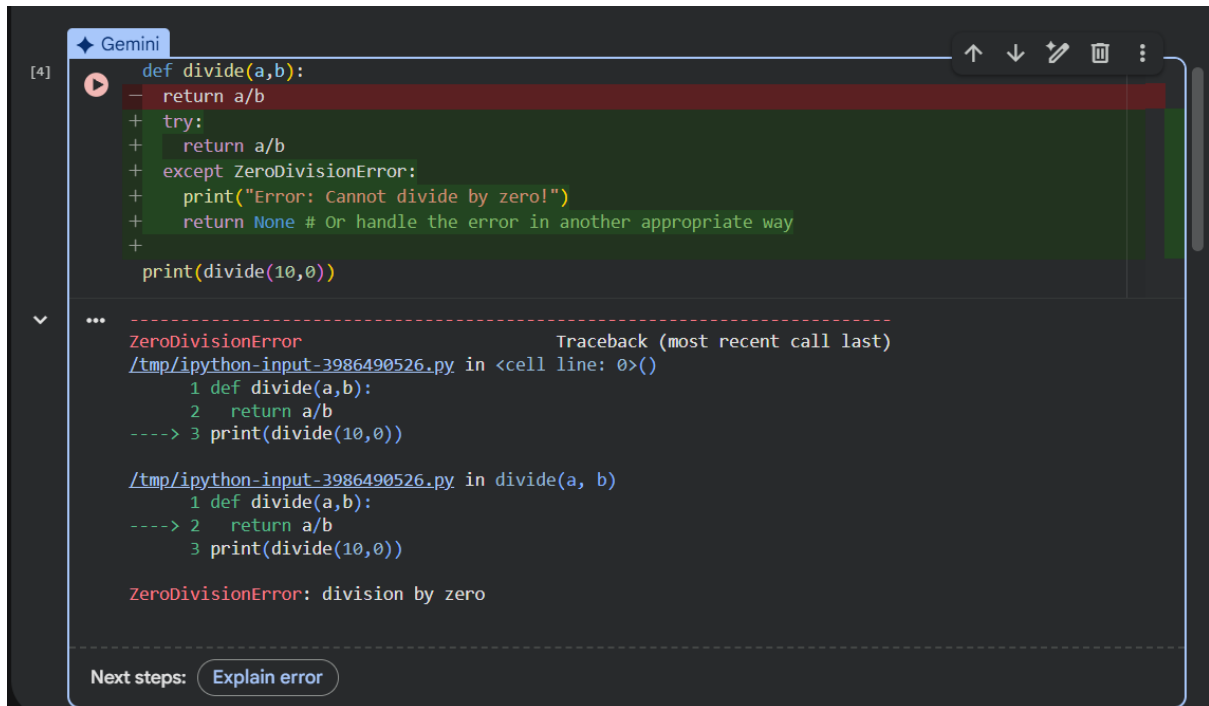
The screenshot shows the output of the corrected `count_down(5)` function. The output is a list of numbers from 5 to 0, which is the correct result of the function.

Code Explanation:

The original `count_down` function had a couple of logical issues:

1. Infinite Loop: Inside the while $n \geq 0$: loop, n was incremented ($n+=1$), meaning n would never become less than 0, resulting in an infinite loop.
2. Stack Overflow: The function recursively called `count_down(5)` within the loop, which would quickly lead to a stack overflow error.

Task 3: Handling Runtime Errors (Division by Zero)



```
[4] def divide(a,b):  
    return a/b  
+   try:  
+       return a/b  
+   except ZeroDivisionError:  
+       print("Error: Cannot divide by zero!")  
+       return None # Or handle the error in another appropriate way  
+  
    print(divide(10,0))  
  
-----  
ZeroDivisionError                                Traceback (most recent call last)  
/tmp/ipython-input-3986490526.py in <cell line: 0>()  
      1 def divide(a,b):  
      2     return a/b  
----> 3 print(divide(10,0))  
  
/tmp/ipython-input-3986490526.py in divide(a, b)  
      1 def divide(a,b):  
----> 2     return a/b  
      3 print(divide(10,0))  
  
ZeroDivisionError: division by zero
```

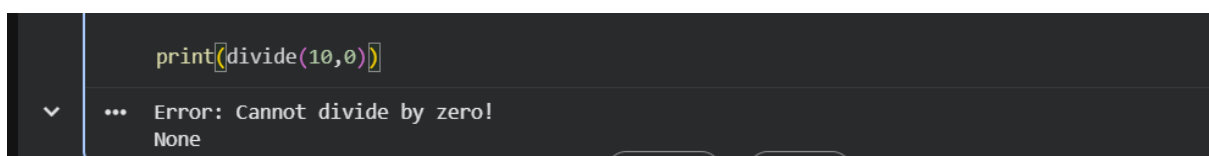
Next steps: [Explain error](#)

Code:



```
[9] def divide(a,b):  
    try:  
        return a/b  
    except ZeroDivisionError:  
        print("Error: Cannot divide by zero!")  
        return None # Or handle the error in another appropriate way  
  
    print(divide(10,0))
```

Output:



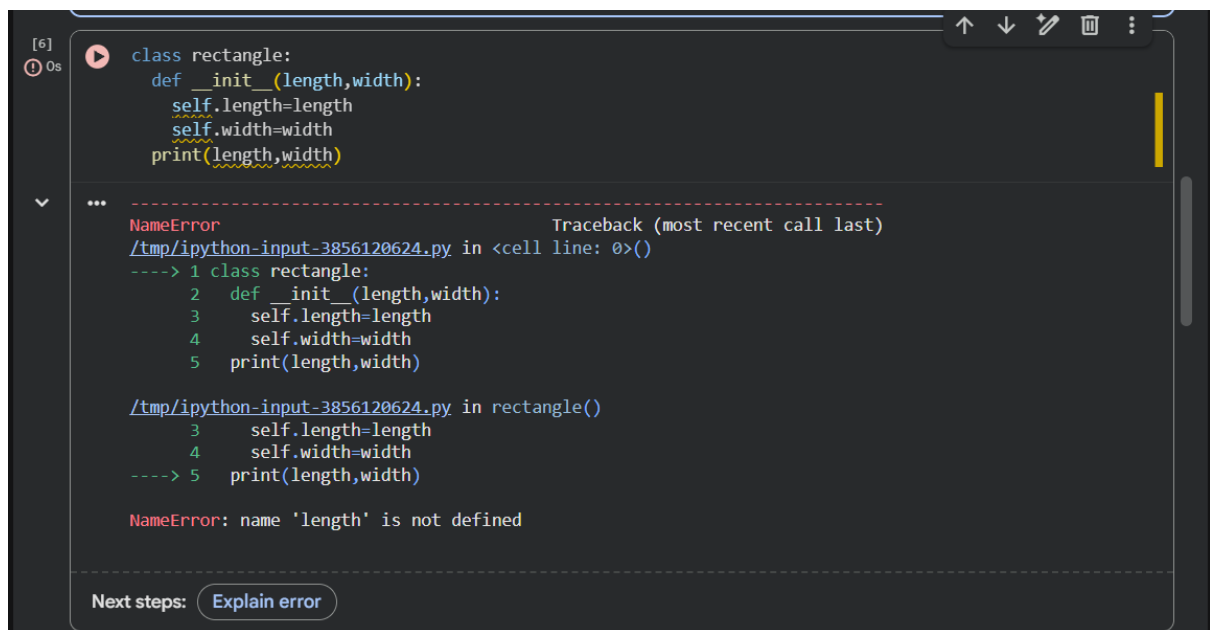
```
print(divide(10,0))  
  
... Error: Cannot divide by zero!  
None
```

Code Explanation:

You're trying to divide by zero, which will cause a runtime error. I've updated the divide function to use a try-except block to catch the ZeroDivisionError and print an informative message instead of crashing. This makes your code more robust.

The divide function in cell mKyXoLbmPsgy executed as expected. Since you attempted to divide by zero, the try-except block caught the ZeroDivisionError, printed "Error: Cannot divide by zero!", and then returned None, which is also printed to the console.

Task 4: Debugging Class Definition Errors



```
[6] 0s
class rectangle:
    def __init__(length,width):
        self.length=length
        self.width=width
        print(length,width)

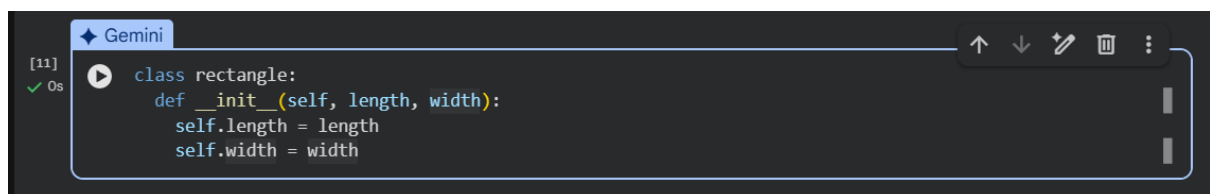
...
NameError                                Traceback (most recent call last)
/tmp/ipython-input-3856120624.py in <cell line: 0>()
----> 1 class rectangle:
      2     def __init__(length,width):
      3         self.length=length
      4         self.width=width
      5         print(length,width)

/tmp/ipython-input-3856120624.py in rectangle()
      3         self.length=length
      4         self.width=width
----> 5         print(length,width)

NameError: name 'length' is not defined

Next steps: Explain error
```

Code:



```
[11] 0s
class rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width
```

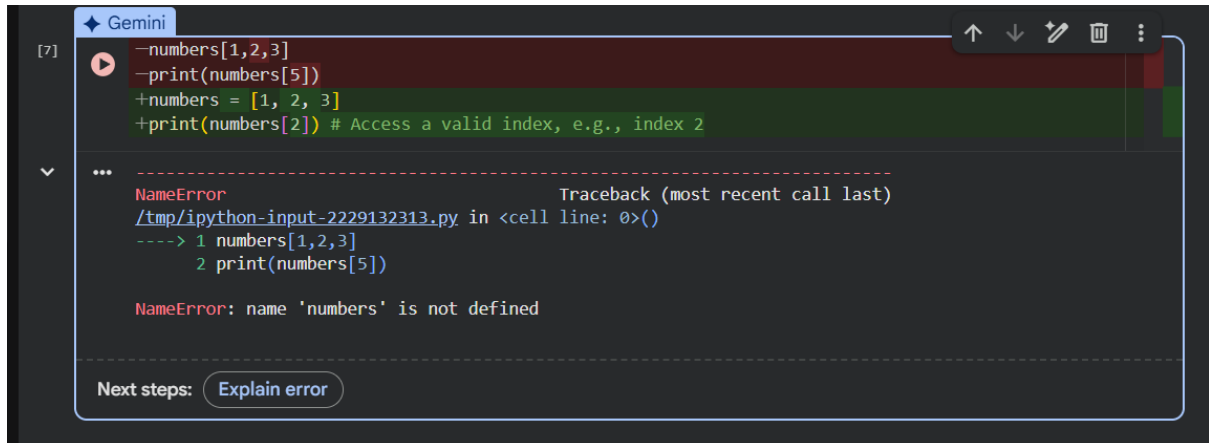
Code explanation:

Encountered a Syntax Error in your rectangle class definition. There are two main issues:

1. Missing Colon: The `__init__` method definition `def __init__(length, width)` is missing a colon (`:`) at the very end. All function and method definitions in Python require a colon.

2. Missing self Parameter: In Python class methods, the first parameter must always be self. This refers to the instance of the object being created. You need to include self in the __init__ signature.

Task 5: Resolving Index Errors in Lists



The screenshot shows a Jupyter Notebook cell with the following code and output:

```
[7] -numbers[1,2,3]
     -print(numbers[5])
     +numbers = [1, 2, 3]
     +print(numbers[2]) # Access a valid index, e.g., index 2
```

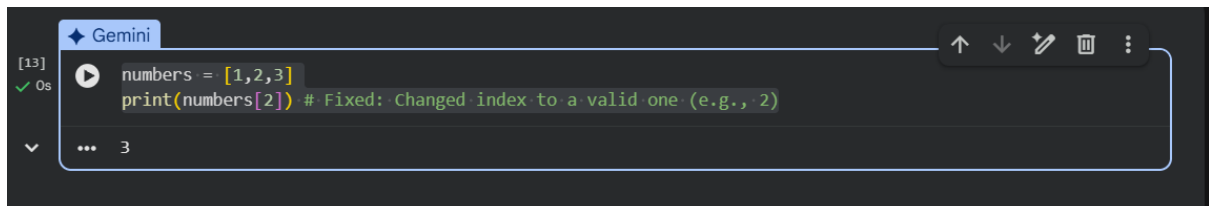
The output shows a `NameError` traceback:

```
-----
NameError                                Traceback (most recent call last)
/tmp/ipython-input-2229132313.py in <cell line: 0>()
----> 1 numbers[1,2,3]
      2 print(numbers[5])

NameError: name 'numbers' is not defined
-----
```

Below the error, there is a "Next steps:" section with a button labeled "Explain error".

Code:



The screenshot shows a Jupyter Notebook cell with the following code and output:

```
[13] numbers = [1,2,3]
     print(numbers[2]) # Fixed: Changed index to a valid one (e.g., 2)
```

The output shows the code executed successfully, with a green checkmark and the number 3.

Code explanation:

The error you're seeing is an Index Error: list index out of range. This happens because your list `numbers` has only 3 elements (at indices 0, 1, and 2), but you're trying to access the element at index 5. To fix this, you need to use an index that is within the valid range of 0 to 2.