

AI Assistant Coding

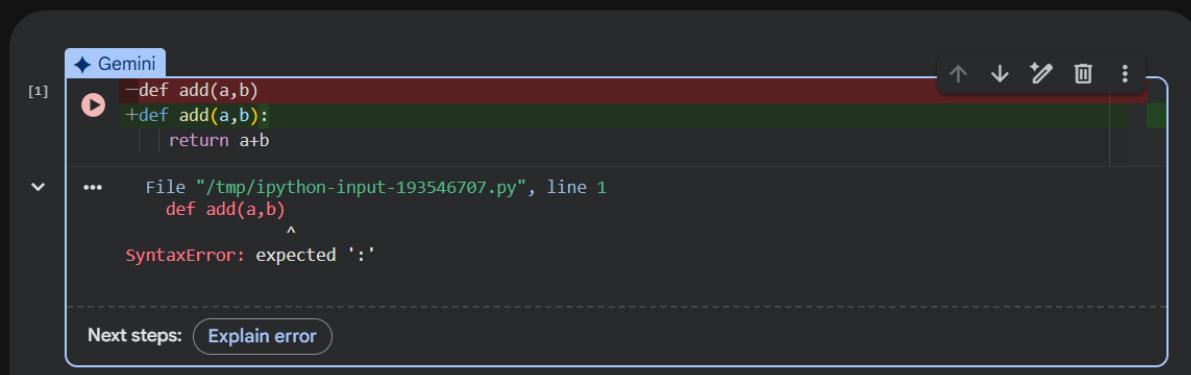
Assignment-7.3

Name: Aman Sarkar

Ht.no: 2303A51273

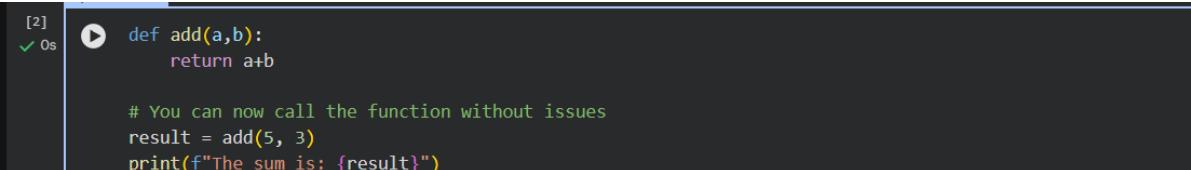
Batch: 05

Task 1: Fixing Syntax Errors



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.

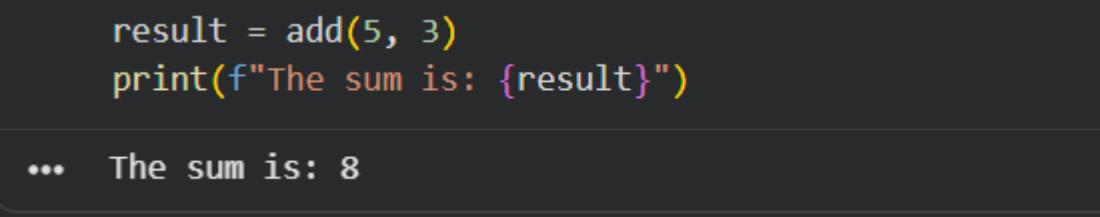
Code:



```
[2] ✓ 0s
▶ 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:



```
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 code editor window with a dark theme. At the top, there's a toolbar with icons for file operations. Below the toolbar, the code editor displays a Python script. The script defines a function `count_down(n)` which prints integers from `n` down to 0. A tooltip is visible over the line `n`, stating: "rupt execution (Ctrl+M I) executed since last change". The code is as follows:

```
[3] def count_down(n):
    while n >= 0:
        print(n)
        n
    print("Execution interrupted at 2:34 PM (0 minutes ago) count_down(5)")

    ...
    5
    6
    7
    8
    9
    10
    11
    12
    13
    14
    15
    16
    17
    18
    19
    20
    21
```

Code:

The screenshot shows a code editor window with a dark theme. The code editor displays a Python script with a green checkmark icon and the text "0s" indicating it's ready to run. The script defines a function `count_down(n)` that prints integers from `n` down to 0. A note in the code indicates a correction: "# Corrected: decrement n to count down". The code is as follows:

```
[8] ✓ 0s def count_down(n):
    while n >= 0:
        print(n)
        n -= 1 # Corrected: decrement n to count down

    # Example usage:
    count_down(5)
```

Output:

The screenshot shows a code editor window with a dark theme. The code editor displays the output of the `count_down(5)` function. The output shows the numbers 5 through 0 printed sequentially. The code is as follows:

```
# Example usage:
count_down(5)

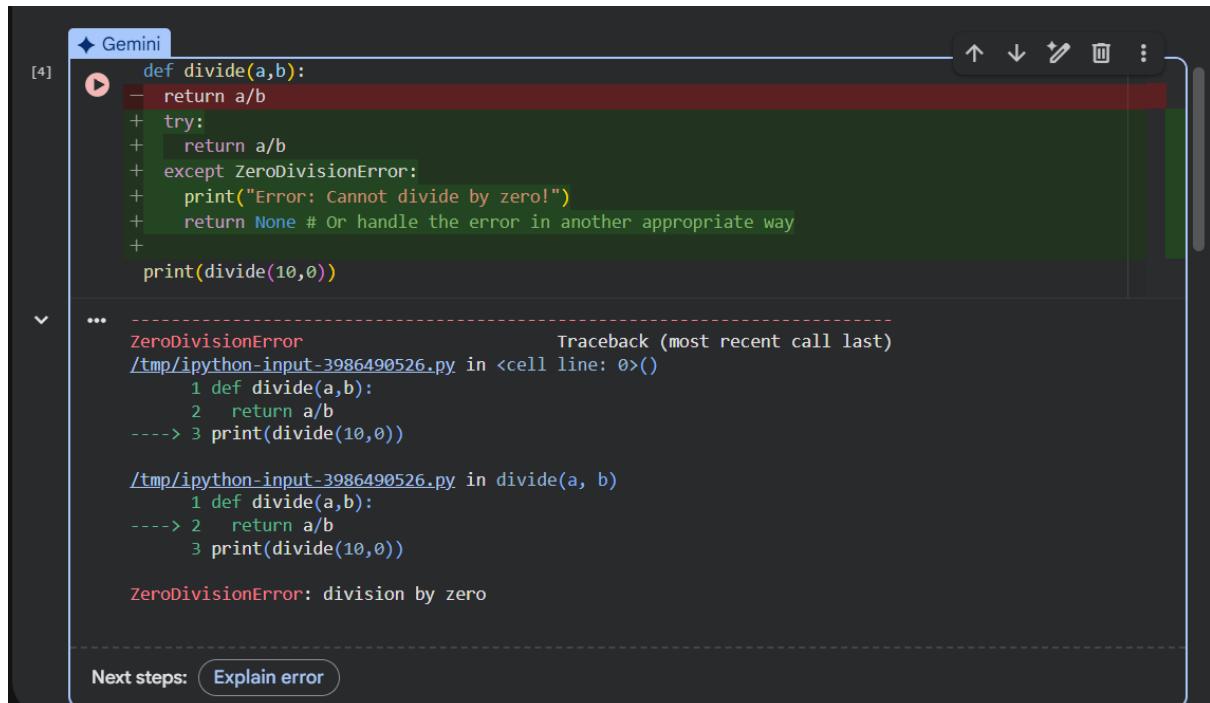
...
5
4
3
2
1
0
```

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)



```

Gemini [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))

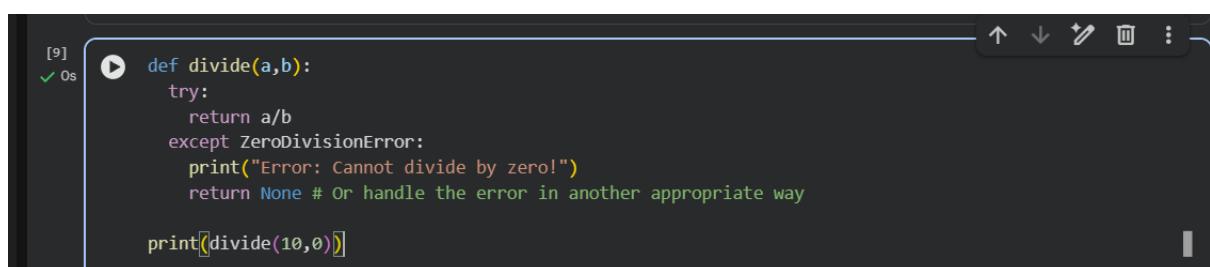
/tmp/ipython-input-3986490526.py in divide(a, b)
      1 def divide(a,b):
----> 2     return a/b
      3 print(divide(10,0))

ZeroDivisionError: division by zero

```

Next steps: Explain error

Code:

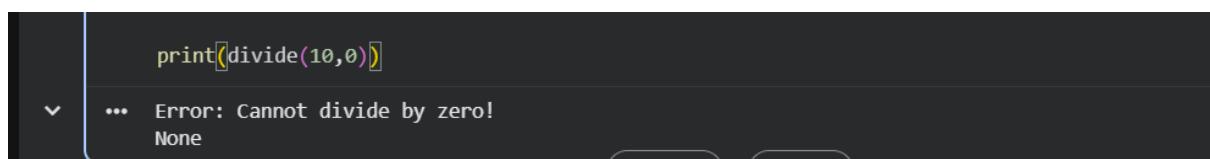


```

[9] 0s
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

A screenshot of a code editor interface. The code shown is:

```
[6] ① 0s ▶ class rectangle:
      def __init__(length,width):
          self.length=length
          self.width=width
          print(length,width)

...
NameError: name 'length' is not defined
```

The code editor has a dark theme. The code is in white text. A red error message 'NameError: name 'length' is not defined' is visible. At the bottom left, there is a button labeled 'Next steps: Explain error'.

Code:

A screenshot of a code editor interface. The code shown is:

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

The code editor has a dark theme. The code is in white text. A blue status bar at the top says 'Gemini'. At the bottom left, there is a button labeled 'Next steps: Explain error'.

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.

- 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 code editor window titled "Gemini". In the code area, there is a single line of Python code: `print(numbers[5])`. Below the code, a red box highlights the index `[5]`. A tooltip or callout box points to this index with the text "Access a valid index, e.g., index 2". The output section shows a traceback for a `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`

Code:

The screenshot shows a code editor window titled "Gemini". The code has been modified to: `numbers = [1,2,3] print(numbers[2])`. A tooltip or callout box points to the index `[2]` with the text "Fixed: changed index to a valid one (e.g., 2)". The output section shows the result of the corrected code: `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.