

# AI Assistant Coding

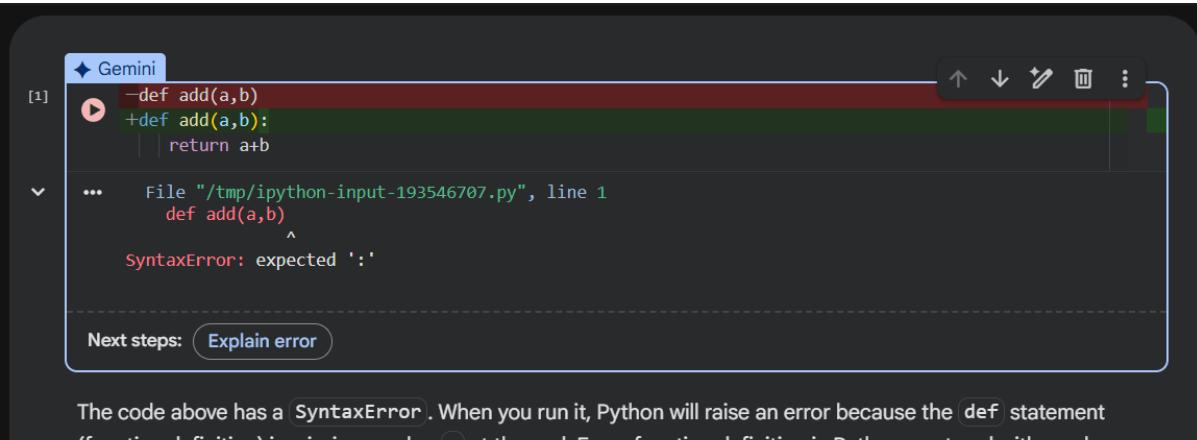
## Assignment-7.3

Name: Saketh Birru

Roll No:2303A51403

Batch:06

### Task 1: Fixing Syntax Errors



The screenshot shows the Gemini AI interface. A code editor window displays the following Python code:

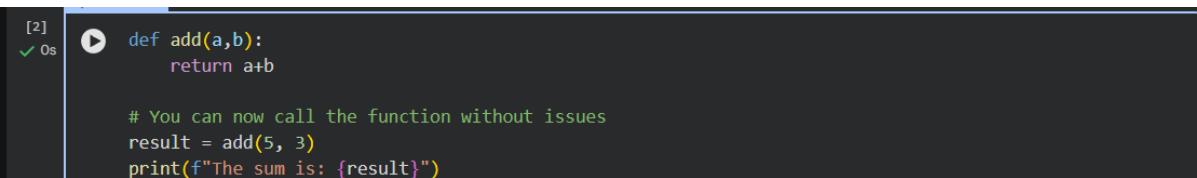
```
[1] Gemini
- def add(a,b)
+def add(a,b):
    return a+b

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

Below the code, a button labeled "Explain error" is visible.

A descriptive message below the code states: "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:

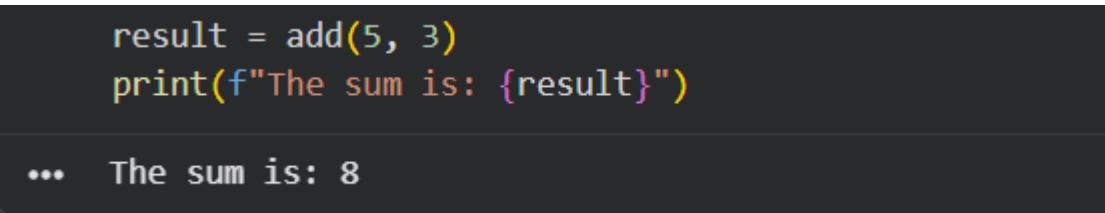


The screenshot shows the Gemini AI interface with the corrected Python code:

```
[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:



The screenshot shows the Gemini AI interface displaying the execution output of the corrected code:

```
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, a status bar displays the number [3] and some execution details: "rupt execution (Ctrl+M I)", "executed since last change", and a timestamp "ted at 2:34 PM (0 minutes ago)". The main area contains a Python script:

```
[3]
def count_down(n):
    while n >= 0:
        print(n)
        n -= 1
    print("Count down finished!")

# Example usage:
count_down(5)
```

The code is intended to print numbers from 5 down to 0. A logical error is present in the loop body: instead of decrementing `n` by 1, it increments it by 1. This causes the loop to run indefinitely. The status bar also indicates that the code has been executed since the last change.

**Code:**

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, a status bar displays the number [8] and a green checkmark icon followed by "0s". The main area contains a Python script:

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

# Example usage:
count_down(5)
```

The code is now corrected. The comment `# Corrected: decrement n to count down` explains the fix. The original increment operation `n += 1` has been replaced with `n -= 1`. The script includes an example usage section at the bottom.

**Output:**

The screenshot shows a code editor window with a dark theme. The main area contains the same Python script as the previous code block, but it is now being run. The output is displayed below the code:

```
# Example usage:
count_down(5)

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

The output shows the numbers 5, 4, 3, 2, 1, and 0 printed sequentially, indicating that the function is working correctly after the fix.

**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)

```

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:**

```

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. There are navigation icons at the top right and a 'Next steps:' button at the bottom.

## Code:

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

```
[11] ✓ 0s ▶ Gemini
      ◆ 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. There are navigation icons at the top right. A blue box highlights the word 'Gemini' in the top left corner.

## Code explanation:

Encountered a SyntaxError 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 pane, there is a single line of Python code: `print(numbers[5])`. Below the code, a red box highlights the index `5` in the list access expression. A tooltip or callout points from this red box to the error message in the output pane. The output pane displays the following traceback:

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

At the bottom of the output pane, there is a button labeled "Explain error".

### Code:

The screenshot shows a code editor window titled "Gemini". In the code pane, there is a line of Python code: `numbers = [1,2,3]`. Below the code, a red box highlights the index `2` in the list assignment expression. A tooltip or callout points from this red box to the explanatory text in the output pane. The output pane displays the following explanatory text:

# Fixed: changed index to a valid one (e.g., 2)

### Code explanation:

The error you're seeing is an IndexError: 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.