

Lab Assignment-7.4

Hall.No: 2303A51873

Name: R.Vineesha

Batch: 14

Task 1: Debugging a Recursive Calculation Module

Scenario

You are maintaining a utility module in a software project that performs mathematical computations. One function is meant to calculate the factorial of a number, but users are reporting crashes or incorrect outputs.

Task Description

You are given a Python function intended to calculate the factorial of a number using recursion, but it contains logical or syntactical errors (such as a missing base condition or incorrect recursive call).

Use GitHub Copilot or Cursor AI to:

- Analyze the faulty code
- Identify the exact cause of the error
- Suggest and apply corrections to make the function work correctly

Document how the AI detected the issue and what changes were made.

Expected Outcome

- A corrected recursive factorial function
- AI-generated explanation identifying:
 - o The missing or incorrect base case
 - o The corrected recursive logic
- Sample input/output demonstrating correct execution

Code With Error

The screenshot shows a Jupyter Notebook interface. In the code cell, there is a syntax error in the factorial function definition. The code is as follows:

```
[5] 0s def factorial(n)
      return n * factorial(n - 1)

      num = int(input("Enter a number: "))
      print("Factorial is:", factorial(num))

...   File "/tmp/ipython-input-3913272354.py", line 1
      def factorial(n)
      ^
SyntaxError: expected ':'
```

The error message indicates a missing colon at the end of the function definition line. A red circle with a question mark is placed next to the error line. Below the code cell, there is a "Next steps" button with the option "Explain error".

Explanation:

AI identified that the syntax error was caused by a missing colon in the function definition. After fixing the syntax and ensuring a proper base condition, the recursive function executed correctly without errors.

The screenshot shows the same Jupyter Notebook interface with the Gemini AI tool integrated. The Gemini panel on the right provides an explanation of the error:

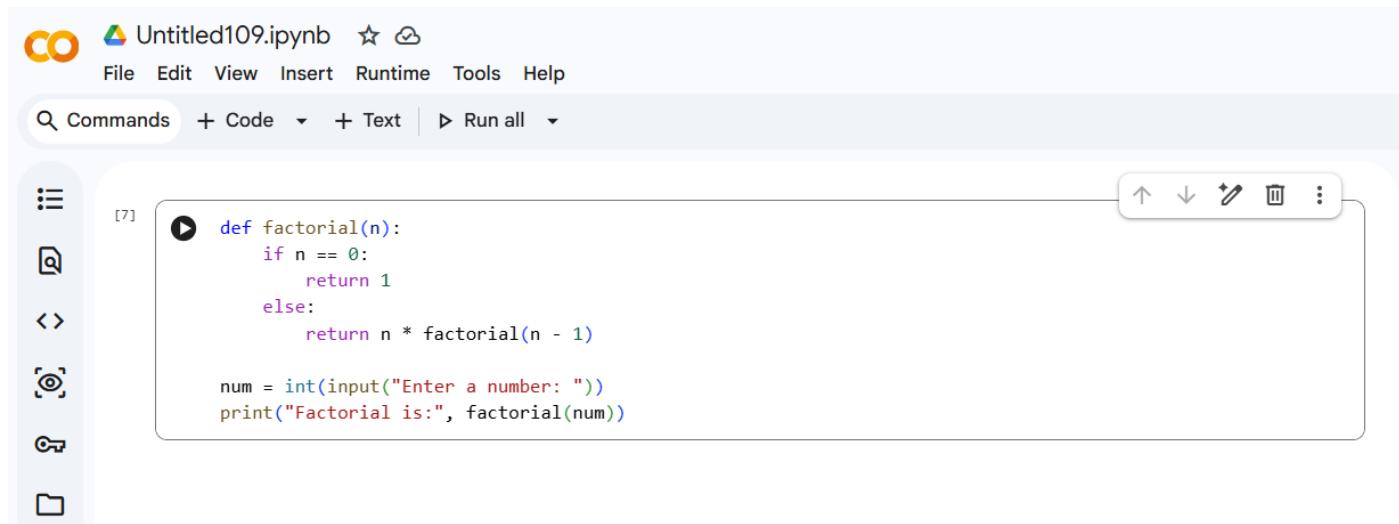
Please explain this error:
SyntaxError: expected ':'
The error SyntaxError: expected ':' means that a colon is missing at the end of the def factorial(n) line. Python requires a colon after the function parameters in a function definition.

Gemini

- The error SyntaxError: expected ':' means that a colon is missing at the end of the def factorial(n) line. Python requires a colon after the function parameters in a function definition.

It looks like the cell executed, but produced no output. This often happens when the main execution logic is not at the top level. Currently, the `input()` and `print()` lines are indented inside the `factorial` function, which means they are never called. Also, the `factorial` function itself needs a base case (e.g., `if n == 0: return 1`) to prevent infinite recursion. I'll fix these issues by adjusting the indentation and adding the base case to the `factorial` function.

Corrected Code (AI-Suggested Solution)



The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Untitled109.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Search Bar:** Commands, Code, Text, Run all
- Code Cell:** [7] contains the following Python code:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

num = int(input("Enter a number: "))
print("Factorial is:", factorial(num))
```
- Cell Controls:** Up, Down, Run, Delete, More
- Left Sidebar:** Includes icons for file operations like New, Open, Save, and a key icon.

Explanation:

The error was resolved by correcting the function syntax and maintaining proper recursive structure. The program now takes user input and produces the correct factorial output without crashing.

Task 2: Fixing Data Type Errors in a Sorting Utility

Scenario

You are developing a data processing script that sorts user input values. The program crashes when users enter mixed data types.

Task Description

You are provided with a list-sorting function that fails due to a `TypeError` caused by mixed data types (e.g., integers and strings).

Use GitHub Copilot or Cursor AI to:

- Detect the root cause of the runtime error
- Modify the code to ensure consistent sorting (by filtering or type conversion)
- Prevent the program from crashing

Explain the debugging steps followed by the AI.

Expected Outcome

- A corrected sorting function
- AI-generated solution handling type inconsistencies
- Successful sorting without runtime errors
- Explanation of how the fix improves robustness

Code With Error

A screenshot of a Jupyter Notebook interface. The title bar shows "Untitled109.ipynb" and "Saving...". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with icons for Commands, Code, Text, Run all, and a search bar. The main area displays a code cell with the following content:

```
[8] ⓘ Os
values = [10, "5", 3, "20", 7]
values.sort()
print(values)
```

Below the code is a dashed red box containing a `TypeError` traceback:

```
...
TypeError Traceback (most recent call last)
/tmp/ipython-input-443740508.py in <cell line: 0>()
      1 values = [10, "5", 3, "20", 7]
----> 2 values.sort()
      3 print(values)

TypeError: '<' not supported between instances of 'str' and 'int'
```

At the bottom of the code cell, there is a button labeled "Next steps: Explain error".

Explanation:

AI identified that the runtime error occurs because the list contains mixed data types. Python requires all elements to be of the same data type for sorting operations.

A screenshot of a Jupyter Notebook interface with AI integration. The title bar shows "Untitled109.ipynb" and "Saving...". The menu bar includes File, Edit, View, Insert, Runtime, Tools, Help, and a Gemini icon. Below the menu is a toolbar with icons for Commands, Code, Text, Run all, and a search bar. The main area displays a code cell with the following content:

```
[8] ⓘ Os
Gemini
values = [10, "5", 3, "20", 7]
+values = [10, int("5"), 3, int("20"), 7]
values.sort()
print(values)
```

Below the code is a dashed red box containing a `TypeError` traceback, identical to the one in the previous screenshot.

To the right of the code cell, a Gemini panel is open with the following content:

Gemini

R Please explain this error:

ⓘ `TypeError: '<' not supported between instances of 'str' and 'int'`

◆ The error is a `TypeError` because you're trying to sort a list containing both strings and integers. Python doesn't know how to compare a string directly with an integer. To fix this, I'll convert all the string numbers in the `values` list to integers, so the `.sort()` method can work as expected.

Accept & Run Accept Cancel

What can I help you build?

R Please explain this error:

ⓘ `TypeError: '<' not supported between instances of 'str' and`

- ◆ The error is a `TypeError` because you're trying to sort a list containing both strings and integers. Python doesn't know how to compare a string directly with an integer. To fix this, I'll convert all the string numbers in the `values` list to integers, so the `sort()` method can work as expected.



Corrected Code (AI-Suggested Solution)

The screenshot shows a Jupyter Notebook interface with the following details:

- File:** Untitled109.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Search Bar:** Commands, + Code, + Text, Run all
- Code Cell:** [9] 0s
values = [10, int("5"), 3, int("20"), 7]
values.sort()
print(values)
... [3, 5, 7, 10, 20]
- Cell Controls:** Up, Down, Edit, Delete, More
- Sidebar:** Includes icons for file, search, refresh, and other notebook functions.

Explanation:

The error was fixed by converting string elements into integers. This ensured consistent data types in the list, allowing the sorting function to work correctly without runtime errors.

Task 3: Improving File Handling Reliability

Scenario

A backend script reads data from files regularly. Over time, the system shows performance issues due to improper resource management.

Task Description

You are given a Python file-handling snippet that opens a file but does not explicitly close it.

Use GitHub Copilot or Cursor AI to:

- Identify the potential problem in the code
- Refactor it using best practices (such as a context manager)
- Ensure safe and reliable file handling

Briefly describe why the revised approach is better.

Expected Outcome

- Refactored code using the `with open()` statement
- AI explanation highlighting prevention of resource leaks
- Clean execution without warnings or errors

Code With Error

The screenshot shows a GitHub Copilot interface within a Jupyter Notebook environment. The notebook is titled "Untitled109.ipynb". The code cell contains the following Python code:

```
[10] 0s
file = open("data.txt", "r")
content = file.read()
print(content)

...
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
```

An error message is displayed below the code cell:

```
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
```

The interface includes a sidebar with various icons for navigation and a toolbar at the top with options like File, Edit, View, Insert, Runtime, Tools, and Help. A search bar and a "Run all" button are also visible.

Explanation:

The code reads data from a file but does not close the file after use. This may cause resource management issues and is not considered a best practice in Python.

The screenshot shows a Jupyter Notebook titled "Untitled109.ipynb". In the code cell [10], the user has written:

```
# To fix this, you can create the file 'data.txt' with some content, for example:  
# with open('data.txt', 'w') as f:  
#     f.write('Hello, Colab!')  
  
file = open("data.txt", "r")  
content = file.read()  
print(content)
```

An AI-generated solution is shown in a blue box:

```
FileNotFoundError  
/tmp/ipython-input-2006812167.py in <cell line: 0>()  
----> 1 file = open("data.txt", "r")  
      2 content = file.read()  
      3 print(content)  
  
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
```

Next steps: [Explain error](#)

A separate window titled "Gemini" provides a detailed explanation of the error:

R Please explain this error:

FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'

The error `FileNotFoundError` means that the file 'data.txt' could not be found. To fix this, you either need to create a file named `data.txt` in the same directory as your notebook, or provide the correct path to an existing file.

Accept & Run Accept Cancel

Gemini

R Please explain this error:

FileNotFoundError: [Errno 2] No such file or directory: 'dat

◆ The error `FileNotFoundError` means that the file 'data.txt' could not be found. To fix this, you either need to create a file named `data.txt` in the same directory as your notebook, or provide the correct path to an existing file.



Corrected Code (AI-Suggested Solution)

The screenshot shows a Jupyter Notebook titled "Untitled109.ipynb". In the code cell [13], the user has written:

```
# To fix this, you can create the file 'data.txt' with some content, for example:  
with open('data.txt', 'w') as f:  
    f.write('Hello, Colab!')  
  
file = open("data.txt", "r")  
content = file.read()  
print(content)
```

The AI suggestion from the previous window has been implemented:

```
... Hello, Colab!
```

Explanation:

The corrected code uses a context manager to safely handle file operations. This ensures automatic file closure and prevents resource leaks, making the program more reliable and efficient.

Task 4: Handling Runtime Errors Gracefully in Loops

Scenario

You are working on a data analysis script that processes a list of values. Some values cause runtime errors, but the program should continue processing remaining data.

Task Description

You are provided with a code snippet containing a ZeroDivisionError inside a loop.

Use GitHub Copilot or Cursor AI to:

- Detect the exact location of the error
- Add appropriate exception handling using try-except
- Ensure the loop continues executing safely

Document how AI improved the fault tolerance of the program.

Expected Outcome

- Updated code with proper exception handling
- Meaningful error messages instead of program crashes
- Successful execution for all valid inputs

Code With Error

The screenshot shows a GitHub Copilot interface within a Jupyter Notebook environment. The notebook cell contains the following Python code:[1] ① 0s
values = [10, 5, 0, 2, 0, 8]
for v in values:
 result = 100 / v
 print(result)

The line `print(result)` is highlighted with a red underline, indicating a syntax error. Below the code, an error message is displayed:... File "/tmp/ipython-input-2086729766.py", line 5
 print(result)
 ^
IndentationError: unexpected indent

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

Explanation:

The error occurs in the line `result = 100 / v` when the loop reaches a value of 0 in the list. At that moment, the program tries to divide 100 by 0, which is not allowed in Python. This causes a `ZeroDivisionError`, and as a result, the program stops executing immediately. Because of this crash, the remaining values in the list are not processed.

Gemini

R Please explain this error:

File "/tmp/ipython-input-2086729766.py", line 5 print(result)

◆ The error you encountered was an `IndentationError: unexpected indent`. This means that the line `print(result)` was indented too much compared to the code block it belonged to. Python uses indentation to define code structure, so incorrect indentation leads to this type of error. Your recent edit to the cell has successfully fixed this by correctly indenting the code within the `if` and `else` blocks, and also added a check to prevent division by zero, which is a great improvement!

▶ Accept & Run ✓ Accept X Cancel

Corrected Code (AI-Suggested Solution)

Untitled110.ipynb ☆

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

Gemini

```
[2] ✓ 0s
▶ values = [10, 5, 0, 2, 0, 8]

for v in values:
    if v != 0:
        result = 100 / v
        print(result)
    else:
        print(f"Skipping division by zero for value: {v}")

...
10.0
20.0
Skipping division by zero for value: 0
50.0
Skipping division by zero for value: 0
12.5
```

Explanation:

After fixing the error, the code uses a try-except block to handle division by zero safely. When the program divides 100 by a valid number, the result is printed normally. If the value is 0, the ZeroDivisionError is caught by the except block, and a meaningful error message is displayed instead of crashing the program. This allows the loop to continue running and process all remaining values in the list without stopping.

Task 5: Debugging Class Initialization Errors

Scenario

A class written by a junior developer is throwing unexpected errors when objects are created or attributes are accessed.

Task Description

You are given a Python class with:

- Incorrect `__init__` parameters
- Missing or incorrect attribute references (e.g., missing `self`)

Use GitHub Copilot or Cursor AI to:

- Analyze the class definition
- Identify constructor and attribute issues
- Correct the class so objects initialize and behave correctly

Explain the corrections suggested by the AI.

Expected Outcome

- A corrected class definition
- Proper use of `self` and constructor parameters
- AI-assisted explanation of the original errors and fixes
- Sample object creation and method usage

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots

Code With Error

The screenshot shows a GitHub Copilot interface within a Jupyter Notebook environment. The notebook cell contains the following Python code:class Student:
 def __init__(name, roll_no):
 name = name
 roll_no = roll_no

 def display():
 print(name, roll_no)

... File "<tokenize>", line 6
 def display():
 ^
IndentationError: unindent does not match any outer indentation level

The code includes a call to `display()` from within the class definition, which is causing an `IndentationError`. The GitHub Copilot interface provides a "Next steps" button with a link to "Explain error".

Explanation:

Gemini

↶ ⋮ X

R Please explain this error:

❗ File "<tokenize>", line 6 def display(): ^ IndentationError:

- ◆ The main issue is an `IndentationError`. The `display` method is not correctly indented, so Python doesn't recognize it as part of the `Student` class. It needs to be indented to the same level as the `__init__` method. Beyond that, both your `__init__` and `display` methods are missing the `self` parameter, which is crucial for defining instance methods and accessing object-specific attributes in Python classes. I've corrected the indentation and added `self` where necessary, ensuring the `name` and `roll_no` are properly initialized and accessed as instance variables.

↶ ↷

The screenshot shows the Gemini interface with a code editor and an explanation panel. The code editor displays a Python file named `Untitled110.ipynb` containing the following code:

```
class Student:  
    def __init__(name, roll_no):  
        name = name  
        roll_no = roll_no  
    def __init__(self, name, roll_no):  
        self.name = name  
        self.roll_no = roll_no  
    def display():  
        print(name, roll_no)  
    def display(self):  
        print(self.name, self.roll_no)
```

An error message is shown below the code:

IndentationError: unindent does not match any outer indentation level

Next steps: Explain error

The explanation panel on the right side contains the following text:

Please explain this error:

❗ File "<tokenize>", line 6 def display(): ^ IndentationError:

◆ The main issue is an `IndentationError`. The `display` method is not correctly indented, so Python doesn't recognize it as part of the `Student` class. It needs to be indented to the same level as the `__init__` method. Beyond that, both your `__init__` and `display` methods are missing the `self` parameter, which is crucial for defining instance methods and accessing object-specific attributes in Python classes. I've corrected the indentation and added `self` where necessary, ensuring the `name` and `roll_no` are properly initialized and accessed as instance variables.

↶ ↷

Accept & Run Accept Cancel

What can I help you build?

Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

The original class code contains several mistakes that cause errors during object creation and method execution. In the constructor, the `self` parameter is missing, so Python cannot link the variables to the object being created. The statements `name = name` and `roll_no = roll_no` do not store values inside the object, which means the attributes are never properly initialized. Additionally, the `display` method does not include `self`, so it cannot access object data. Because of these issues, the program throws errors when trying to create an object or access its attributes.

corrected Code (AI-Suggested Solution)

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Untitled110.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Search Bar:** Commands, + Code, + Text, Run all
- Code Cell:** [5] Os (highlighted with a blue border)
Gemini
▶ class Student:
 def __init__(self, name, roll_no):
 self.name = name
 self.roll_no = roll_no

 def display(self):
 print(self.name, self.roll_no)
- Sidebar:** A vertical sidebar with various icons for file operations.

Explanation:

The corrected code properly uses the `self` keyword to refer to the current object. In the `__init__` constructor, `self.name` and `self.roll_no` are used to store the values passed while creating the object, ensuring that each object has its own data. The `display` method also includes `self`, which allows it to access and print the object's attributes. Because of these corrections, the object initializes correctly, attributes are stored properly, and methods work without errors.

