

AI-assisted coding

Assignment -7.4

Name: V. Vyshnavi

Batch-14

HT.No.: 2303A51968

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:
 - The missing or incorrect base case
 - The corrected recursive logic
- Sample input/output demonstrating correct execution

Code with error:

The screenshot shows a Jupyter Notebook interface. The file is titled "Untitled38.ipynb". The code cell contains the following Python code:

```
[2] ① Os
def factorial(n)
    return n * factorial(n-1)

... File "/tmp/ipython-input-4116075141.py", line 1
    def factorial(n)
               ^
SyntaxError: expected ':'
```

A red circle with a question mark icon is next to the first line of code, indicating an error. A tooltip below the code cell says "Next steps: Explain error".

Explanation:

! `SyntaxError: expected ':'`

◆ The `SyntaxError: expected ':'` indicates that the function definition `def factorial(n)` is missing a colon at the end. In Python, all function definitions must end with a colon. I've corrected this for you.

[Like](#) [Unlike](#)

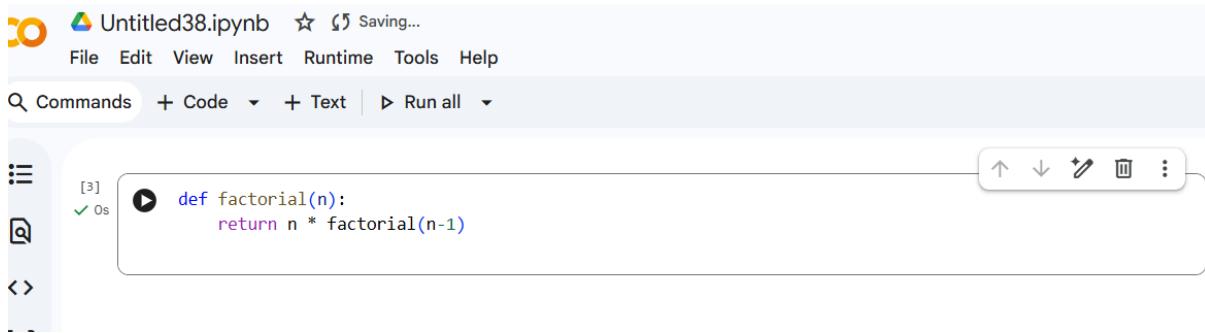
[Explain the corrected factorial function](#) [Execute the factorial function with an example input](#)

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](#)

Actual code:



The screenshot shows a Jupyter Notebook interface. The title bar says "Untitled38.ipynb" and "Saving...". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with search, command, code, text, run all, and other options. On the left is a sidebar with icons for file, cell, and other notebook functions. A code cell is selected, containing the following Python code:

```
[3]  def factorial(n):
        return n * factorial(n-1)
```

To the right of the code cell is a toolbar with up, down, edit, delete, and more options.

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:

The screenshot shows a Jupyter Notebook interface with a single code cell containing the following Python code:

```
data = [10, "20", 5, "apple", 15]
print(sorted(data))
```

The cell has a status bar indicating "[4] 0s". The output area shows a red error icon followed by the error message:

```
... ---->
TypeError: 'less than' not supported between instances of 'str' and 'int'
```

Below the error message, there is a link "Next steps: Explain error".

Explanation:

A user interface for explaining errors. A question "D Please explain this error:" is displayed. Below it is an error message: "TypeError: 'less than' not supported between instances of 'str' and 'int'". A blue diamond icon points to this message with the text "The error TypeError: 'less than' not supported between instances of". At the bottom of the interface are two buttons: "Explain the 'TypeError' in the last cell" and "Show the contents of 'numeric_data' before sorting".

Actual code:

The screenshot shows a Jupyter Notebook interface with the title "Untitled38.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with search, command, code, text, and run all buttons. On the left is a sidebar with various icons. A code cell is selected, containing the following Python code:

```
[5] data = [10, "20", 5, "apple", 15]

numeric_data = []
for item in data:
    try:
        numeric_data.append(int(item))
    except (ValueError, TypeError):
        pass

print(sorted(numeric_data))

... [5, 10, 15, 20]
```

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 Jupyter Notebook interface. The title bar says "Untitled38.ipynb". The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. Below the menu is a toolbar with search, command, code, text, and run all buttons. On the left is a sidebar with icons for file operations. A code cell [6] contains the following Python code:

```
file = open("data.txt", "r")
content = file.read()
print(content)
```

The cell has a red play button icon and a status of "0s". Below the code is a dashed line, followed by a stack trace:

```
... -----
FileNotFoundError                         Traceback (most recent call last)
/ttmp/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'
```

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

Explanation:

A blue circular icon with a white letter "D" is followed by the text "Please explain this error:". Below it is a red circular icon with an exclamation mark, followed by the error message "FileNotFoundError: [Errno 2] No such file or directory: 'data.t'".

To the left of the error message is a blue diamond icon with a white plus sign, followed by the text "The FileNotFoundError: [Errno 2] No such file or directory:".

Below these are three buttons: "Run the code cell again", "Explain why the FileNotFoundError might still be present", and "Show r".

At the bottom of the box, there is a question "What can I help you build?" followed by a plus sign and the text "Gemini 2.5 Flash ▶".

At the very bottom, a note says "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".

Actual code:

The screenshot shows the GitHub Copilot interface for a Jupyter Notebook titled "Untitled38.ipynb". The notebook has one cell, indicated by the number [1] and the text "0s". The cell contains the following Python code:

```
with open("data.txt", "w") as f:  
    f.write("This is some sample data.\n")  
    f.write("Another line of text.")  
  
file = open("data.txt", "r")  
content = file.read()  
print(content)  
file.close()  
  
... This is some sample data.  
Another line of text.
```

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 Jupyter Notebook interface with the following details:

- File:** Untitled38.ipynb
- Cell Number:** [8]
- Code:**

```
[8] 0s
▶ numbers = [10, 5, 0, 20]
for n in numbers:
    print(100 / n)

...
10.0
20.0

ZeroDivisionError
Traceback (most recent call last)
/tmp/ipython-input-1634850788.py in <cell line: 0>()
      2
      3 for n in numbers:
----> 4     print(100 / n)

ZeroDivisionError: division by zero
```
- Output:** The output shows the first two values (10.0 and 20.0) followed by an error message: `ZeroDivisionError: division by zero`.
- Next steps:** A link labeled "Explain error" is visible.

Explanation:

The screenshot shows the Gemini AI interface with the following details:

- Task Type:** D (Documentation)
- Question:** Please explain this error:
- Answer:** `! ZeroDivisionError: division by zero`
- Explanation:** The `ZeroDivisionError` occurs because your code tries to divide 100 by
- Buttons:** Explain why 'data.txt' is still not found, Provide code to create and read 'data.txt', Check
- Feedback:** What can I help you build? + Gemini 2.5 Flash ▶
- Note:** Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Actual code:

A screenshot of a Jupyter Notebook interface. The title bar says "Untitled38.ipynb". The code cell contains the following Python code:

```
numbers = [10, 5, 0, 20]

for n in numbers:
    if n == 0:
        print("Cannot divide by zero!")
    else:
        print(100 / n)
```

The output pane shows the results of running the code:

```
... 10.0
20.0
Cannot divide by zero!
5.0
```

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

Code with error:

A screenshot of a Jupyter Notebook interface. The title bar says "Untitled38.ipynb". The code cell contains the following Python code:

```
class Student:
    def __init__(name, age):
        name = name
        age = age

    def display():
        print("Name:", name, "Age:", age)
```

The output pane shows the error message:

```
... File "/tmp/ipython-input-333388354.py", line 6
      def display()
                           ^
SyntaxError: expected ':'
```

Explanation:

D Please explain this error:

! `ZeroDivisionError: division by zero`

- ◆ The `ZeroDivisionError` occurs because your code tries to divide 100 by 0 when `n` is 0, which is not allowed. To resolve this, I've added a check: if

[Explain why 'data.txt' is still not found](#)

[Provide code to create and read 'data.txt'](#)

[Check](#)

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](#)

Actual code:

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

- Title:** Untitled38.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Search Bar:** Q Commands + Code + Text | Run all
- Code Cell:** [12] 0s
- Code Content:**

```
class Student:  
    def __init__(self, name, age):  
        self.name = name  
        self.age = age  
  
    def display(self):  
        print("Name:", self.name, "Age:", self.age)
```
- Cell Controls:** Up, Down, Edit, Delete, More