

Assignment-7.4

Task01: 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:

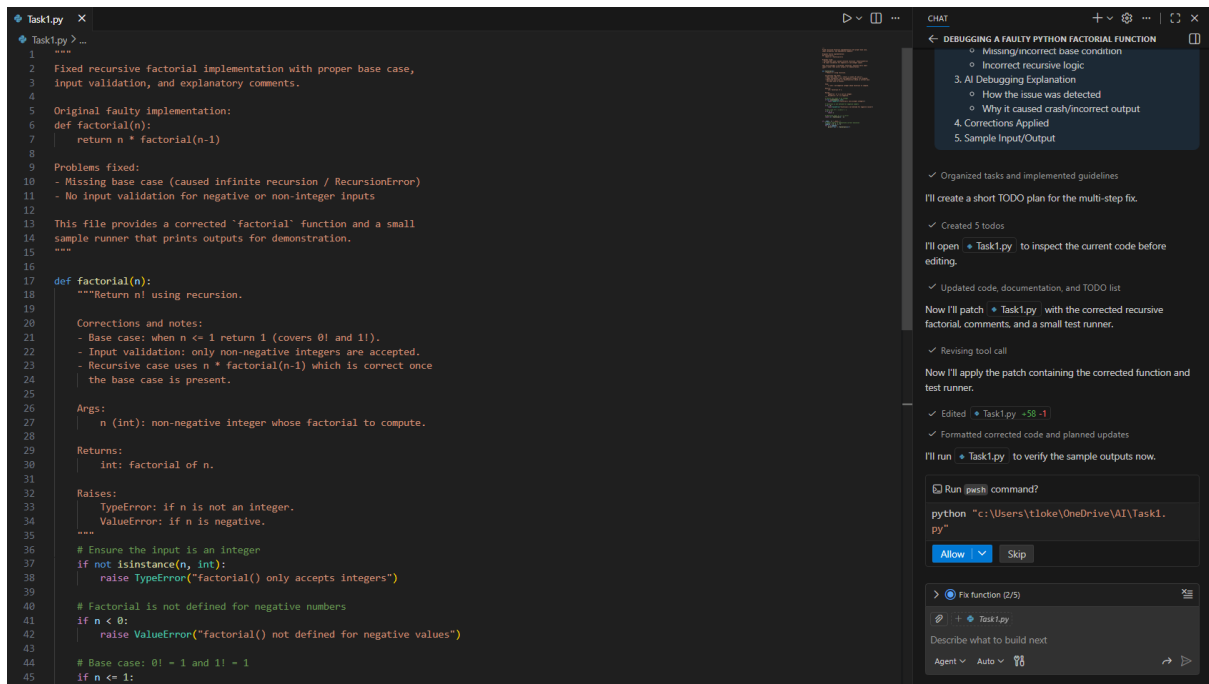
- Analyze the faulty code
- Identify the exact cause of the error
- Suggest and apply corrections to make the function work

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

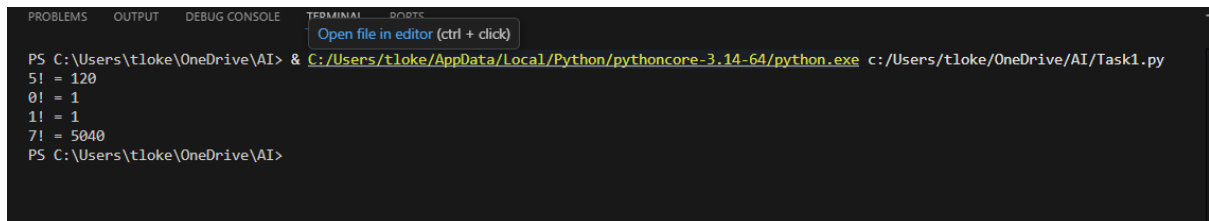
Expected Outcome

- 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



OUTPUT:



TASK-02:

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 Type Error 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

```

1  """
2  Robust sorting helper that handles mixed data types safely.
3
4  Original faulty implementation:
5  def sort_values(data):
6      return sorted(data)
7
8  Problem: When 'data' contains items with types that are not directly
9  comparable (e.g., int and str), Python raises a 'TypeError' while
10 trying to compare elements during sorting.
11
12 This file provides a safer 'sort_values' implementation that defines
13 a deterministic key for sorting mixed types so the function will never
14 raise a TypeError for common Python built-ins.
15 """
16
17 from typing import Iterable, List, Any, Tuple
18
19
20 def _sort_key(x: Any) -> Tuple[int, Any]:
21     """Return a tuple key used to compare mixed-type values safely.
22
23     Key structure (type_priority, comparable_value) where:
24     - type_priority: integer grouping by broad type (numbers first,
25       then strings, then other types). Lower priority sorts earlier.
26     - comparable_value: a value of a consistent, comparable type used
27       for ordering within the group.
28
29     This avoids direct comparisons between incompatible types like
30     int and str which raise TypeError in Python 3.
31     """
32     # Numbers (int, float) - treat all as floats for ordering
33     if isinstance(x, (int, float)):
34         return (0, float(x))
35
36     # Strings: attempt to detect numeric strings and treat them as numbers
37     if isinstance(x, str):
38         # Try numeric conversion (so '10' sorts before '2' numerically)
39         try:
40             return (0, float(x))
41         except ValueError:
42             # Non-numeric strings sort after numbers; use lowercase for
43             # case-insensitive lexicographic ordering
44             return (1, x.lower())
45

```

DEBUGGING A FAULTY PYTHON FACTORIAL FUNCTION

- Corrected Python Code
- Error Explanation
 - Root cause of TypeError
 - Why mixed types cause failure
- AI Debugging Process
 - How the AI detected the issue
 - How the solution was chosen
- Fix Applied
 - Type filtering OR type conversion logic
- Robustness Improvement Explanation
- Sample Input/Output Demonstration

Planned Todo List tasks and updated Task2.py

I'll add a short TODO, inspect `Task2.py`, then patch it to handle mixed-type lists safely.

Managed and tracked todo items

Progress: `Task2.py` is empty. Next I'll write a robust `sort_values` implementation and add demo tests.

Edited `Task2.py` +91 -1

Implement safe sort (2/5)

1 file changed +91 -1

Describe what to build next

Agent Auto 98

OUTPUT:

```

Demo 1: input = [3, '1', 2, '10', 'apple', 'Banana', 0.5]
        sorted = [0.5, '1', 2, 3, '10', 'apple', 'Banana']

Demo 2: input = ['a', 'B', 'c', 2, 1, '10']
        sorted = [1, 2, '10', 'a', 'B', 'c']

Demo 3: input = [None, 5, '3', {'k': 'v'}, (2,)]
        sorted = ['3', 5, (2,), None, {'k': 'v'}]

Demo 4: input = ['100', '20', 3, 4.5, 'abc']
        sorted = [3, 4.5, '20', '100', 'abc']

PS C:\Users\tloke\OneDrive\AI>

```

Task-03: 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

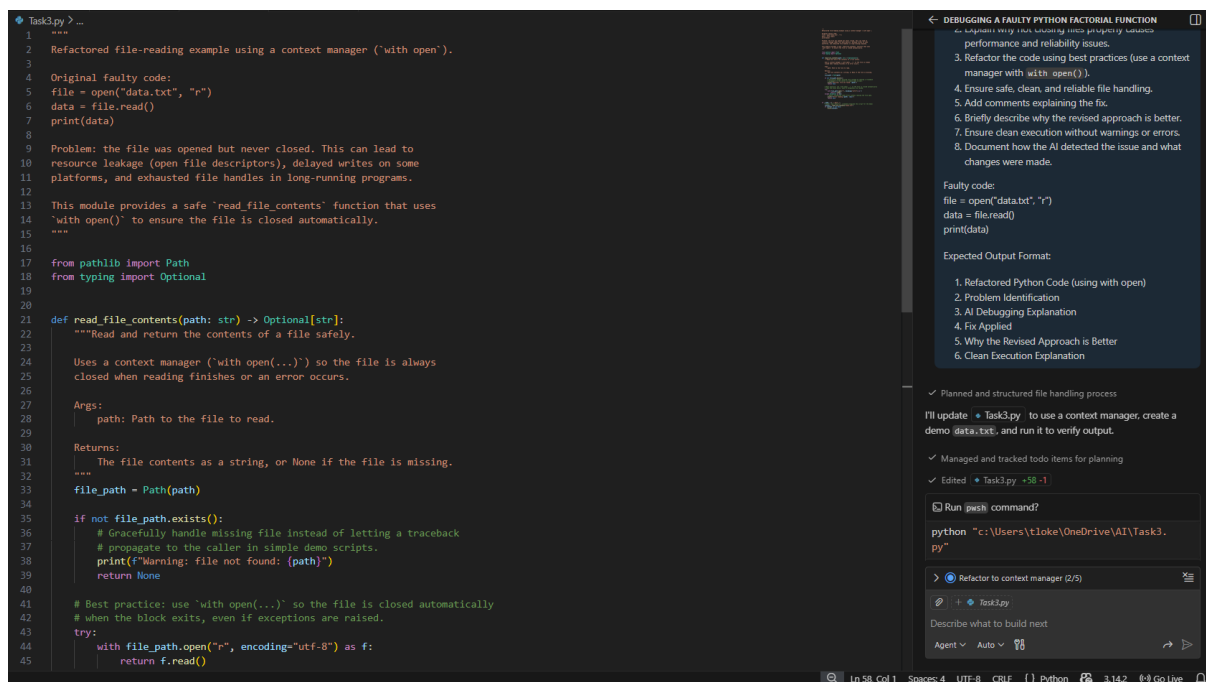
You are given a Python file-handling snippet that opens a file but does not explicitly close it. Use GitHub Copilot or Cursor AI to:

- 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



```
Task3.py > ...
1  """
2  Refactored file-reading example using a context manager ('with open').
3
4  Original faulty code:
5  file = open("data.txt", "r")
6  data = file.read()
7  print(data)
8
9  Problem: the file was opened but never closed. This can lead to
10 resource leakage (open file descriptors), delayed writes on some
11 platforms, and exhausted file handles in long-running programs.
12
13 This module provides a safe 'read_file_contents' function that uses
14 'with open()' to ensure the file is closed automatically.
15 """
16
17 from pathlib import Path
18 from typing import Optional
19
20
21 def read_file_contents(path: str) -> Optional[str]:
22     """Read and return the contents of a file safely.
23
24     Uses a context manager ('with open(...)') so the file is always
25     closed when reading finishes or an error occurs.
26
27     Args:
28         path: Path to the file to read.
29
30     Returns:
31         The file contents as a string, or None if the file is missing.
32     """
33     file_path = Path(path)
34
35     if not file_path.exists():
36         # Gracefully handle missing file instead of letting a traceback
37         # propagate to the caller in simple demo scripts.
38         print(f"Warning: file not found: {path}")
39         return None
40
41     # Best practice: use 'with open(...)' so the file is closed automatically
42     # when the block exits, even if exceptions are raised.
43     try:
44         with file_path.open("r", encoding="utf-8") as f:
45             return f.read()
```

DEBUGGING A FAULTY PYTHON FACTORIAL FUNCTION

1. Refactored Python Code (using with open)

2. Problem Identification

3. AI Debugging Explanation

4. Fix Applied

5. Why the Revised Approach is Better

6. Clean Execution Explanation

Planned and structured file handling process

I'll update Task3.py to use a context manager, create a demo data.txt, and run it to verify output.

Managed and tracked todo items for planning

Edited Task3.py +58 -1

Run bash command?

```
python "c:\Users\tloke\OneDrive\AI\Task3.py"
```

Refactor to context manager (2/5)

Task3.py

Describe what to build next

Agent Auto

```
data.txt
1 Sample data file for Task3 demo.
2 Line 2: Hello, world!
3
```

OUTPUT:

```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/Task3.py
Sample data file for Task3 demo.
Line 2: Hello, world!
Open file in editor (ctrl + click)
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/Task3.py
Sample data file for Task3 demo.
Line 2: Hello, world!
```

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

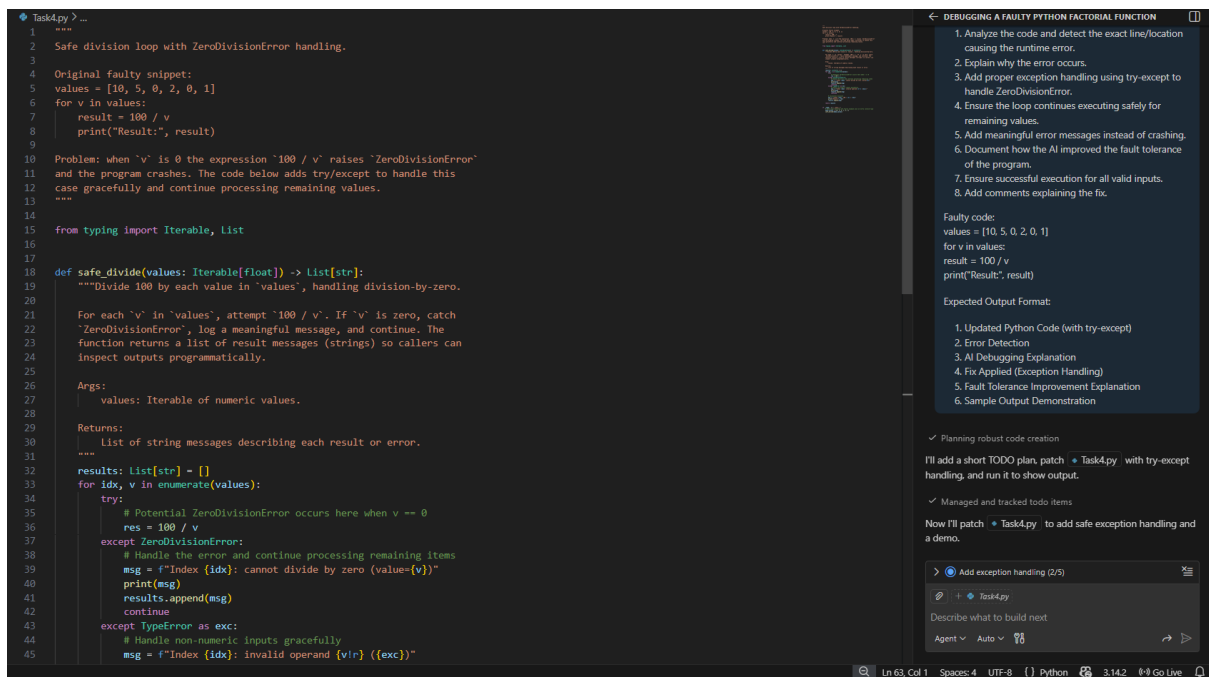
You are provided with a code snippet containing a `ZeroDivisionError` inside a loop. Use GitHub Copilot or Cursor AI to:

- 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



OUTPUT:

```

PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/Task4.py
Result: 10.0
Result: 20.0
Index 2: cannot divide by zero (value=0)
Result: 50.0
Index 4: cannot divide by zero (value=0)
Result: 100.0
PS C:\Users\tloke\OneDrive\AI>

```

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

The screenshot shows a code editor with a Python file named `Task5.py`. The code defines a `Student` class with an `__init__` method and a `display` method. The `__init__` method is corrected to use `self` and properly assign attributes. The `display` method is corrected to use `self` to access instance attributes. The code also includes a docstring explaining the fixes and a small demo showing object creation and method usage.

The sidebar on the right, titled "DEBUGGING A FAULTY PYTHON FACTORIAL FUNCTION", provides a detailed explanation of the fixes:

1. Analyze the class definition and identify all constructor and attribute errors.
2. Explain why these errors occur during object initialization and usage.
3. Correct the class so objects initialize and behave correctly.
4. Ensure proper use of `self` and constructor parameters.
5. Add comments in the code explaining each fix.
6. Document the AI debugging process and reasoning.
7. Explain how the corrections improve class reliability and usability.
8. Provide sample object creation and method usage.
9. Show sample output demonstrating correct behavior.

The sidebar also shows the "Faulty code" and the "Expected Output Format":

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

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

Expected Output Format:

1. Corrected Class Definition
2. Error Identification
3. AI Debugging Explanation
4. Corrections Applied
5. Reliability Improvement Explanation
6. Sample Object Creation & Method Usage
7. Sample Output

OUTPUT:

```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/Task5.py
Created students:
Student(name='Alice', age=20)
Student(name='Bob', age=22)

Display details:
Name: Alice
Age: 20
Name: Bob
Age: 22

Birthday update:
Happy birthday, Alice! You are now 21.
Name: Alice
Age: 21
PS C:\Users\tloke\OneDrive\AI>
```