

Assignment 7.4

Name: N.Akhil

Ht.no: 2303A51065

Bt.no : 16

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).

The screenshot shows a dark-themed instance of Visual Studio Code. In the center is the code editor with the file `ASS7_4.py` open. The code defines a recursive factorial function:`def fact(n):
 """Return n * fact(n-1)
 --> return n! for non-negative integer n using recursion."""
 if n == 0 or n == 1:
 return 1
 return n * fact(n - 1)`The code has a red underline under the line `if n == 0 or n == 1:`, indicating a potential error. To the right of the editor is the AI-assisted Chat panel, which has a title "RECURSION BUG FIX AND EXPLANATION". It contains the following text:

I'll update the factorial function in `ASS7_4.py` to add the missing base case and then briefly explain what went wrong with the original recursion.

✓ Edited `ASS7_4.py` +4 -1

- The bug was that `fact` had no base case: it always called `fact(n-1)` and never stopped. For any input, this leads to infinite recursion and eventually a `RecursionError` (stack overflow).
- A correct recursive factorial must have a stopping condition, e.g. `n = 1` and `1! = 1`. Once it reaches that, it returns a value instead of calling itself again.
- I fixed `ASS7_4.py` so that:
 - If `n` is 0 or 1, it returns 1.
 - Otherwise, it returns `n * fact(n - 1)`, which now eventually reaches the base case and terminates.

At the bottom of the Chat panel, there is a summary: "1 file changed +4 -1" and a "Keep" button. Below the Chat panel is a status bar showing "Ln 6, Col 1 Spaces: 4 UTF-8 Python 3.13.1 Go Live Windsurf (...) Prettier".

- The bug was that `fact` had no base case: it always called `fact(n-1)` and never stopped. For any input, this leads to infinite recursion and eventually a `RecursionError` (stack overflow).

- A correct recursive factorial must have a stopping condition, e.g. $0! = 1$ and $1! = 1$. Once it reaches that, it returns a value instead of calling itself again.

The screenshot shows a VS Code interface with the following details:

- Editor:** The file `ASS7_4.py` is open, containing the following code:

```

1 def fact(n):
2     """Return n! for non-negative integer n using recursion."""
3     if n == 0 or n == 1:
4         return 1
5     return n * fact(n - 1)
6 n=int(input());
7 print(fact(n));

```
- Terminal:** Shows the command `python -u "c:\Users\akhil\Desktop\AI-ASSISTED\ASS7_4.py"` and its output:

```

5
120

```
- Chat:** A GPT-5.1 AI assistant is providing feedback:
 - I'll update the factorial function in `ASS7_4.py` to add the missing base case and then briefly explain what went wrong with the original recursion.
 - ✓ Edited `ASS7_4.py` +4 -1
 - The bug was that `fact` had no base case: it always called `fact(n-1)` and never stopped. For any input, this leads to infinite recursion and eventually a `RecursionError` (stack overflow).
 - A correct recursive factorial must have a stopping condition, e.g. $0! = 1$ and $1! = 1$. Once it reaches that, it returns a value instead of calling itself again.
 - I fixed `ASS7_4.py` so that:
 - If `n` is 0 or 1, it returns 1.
 - Otherwise, it returns `n * fact(n - 1)`, which now eventually reaches the base case and terminates.
- Status Bar:** Shows "GPT-5.1 • 1x".

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).

The screenshot shows the VS Code interface with an AI-assisted feature active. The code editor has the following content:

```
ASST7_4.py
1 arr = [5, "3", 1, "10"]
2 print(sorted(arr))
3
4 def safe_sort_key(value):
5     """Return a key that allows mixed-type sorting safely.
6
7     - Tries to interpret values as numbers (ints/floats, including numeric strings).
8     - Falls back to string representation for non-numeric values.
9     """
10
11     try:
12         # First, try numeric comparison so numeric strings and numbers align
13         return (0, float(value))
14     except (TypeError, ValueError):
15         # Fallback: group after numeric values, sort lexicographically
16         return (1, str(value))
17
18 sorted_arr = sorted(arr, key=safe_sort_key)
19 print(sorted_arr)
20
```

The AI panel on the right suggests handling mixed types and sort safely, mentioning the creation of a `safe_sort_key` function. It also notes that the script will sort by numeric value using this function, which handles both integers and numeric strings without raising errors.

Using a `safe_sort_key` that tries `float(value)` for numeric ordering of ints and numeric strings.

Falling back to `(1, str(value))` for non-numeric values so the sort never raises a `TypeError`.

Sorting with `sorted(arr, key=safe_sort_key)` and printing the result.

The screenshot shows the VS Code interface with the terminal tab active. The terminal output is as follows:

```
ASST7_4.py > safe_sort_key
1 arr = [5, "3", 1, "10"]
2 def safe_sort_key(value):
3     """Return a key that allows mixed-type sorting safely.
4     - Tries to interpret values as numbers (ints/floats, including numeric strings).
5     - Falls back to string representation for non-numeric values.
6     """
7     try:
8         # First, try numeric comparison so numeric strings and numbers align
9         return (0, float(value))
10    except (TypeError, ValueError):
11        # Fallback: group after numeric values, sort lexicographically
12        return (1, str(value))
13 sorted_arr = sorted(arr, key=safe_sort_key)
14 print(sorted_arr)

Microsoft Windows [Version 10.0.26200.7705]
(c) Microsoft Corporation. All rights reserved.

C:\Users\akhil\Desktop\AI-ASSISTED>python -u "c:/Users/akhil/Desktop/AI-ASSISTED\ASST7_4.py"
[1, '3', 5, '10']

C:\Users\akhil\Desktop\AI-ASSISTED>
```

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.

The screenshot shows a Microsoft Windows terminal window and a VS Code interface. The terminal shows the command `python -u "c:/users/akhil/Desktop/AI-ASSISTED\ASS7_4.py"` being run, which results in a `FileNotFoundError` because the file `data.txt` does not exist. The VS Code editor has an open file named `ASS7_4.py` containing the following code:

```
f = open("data.txt", "r")
data = f.read()
with open("data.txt", "r", encoding="utf-8") as f:
    data = f.read()
```

The right-hand sidebar displays AI-assisted refactoring suggestions. It suggests using a `with` context manager and specifying `encoding="utf-8"`. The result of applying these changes is shown in the code editor:

```
with open("data.txt", "r", encoding="utf-8")
    data = f.read()
```

The AI also suggests creating a `data.txt` file in the workspace root so the script can read from it.

Use a `with` context manager so the file is automatically closed.

Specify `encoding="utf-8"` for more predictable text handling

The screenshot shows the same environment as before, but now the `data.txt` file exists in the workspace. The terminal shows the script running successfully and printing "Hello World". The VS Code editor shows the refactored code:

```
with open("data.txt", "r", encoding="utf-8") as f:
    data = f.read()
    print(data)
```

The AI sidebar still suggests creating a `data.txt` file, but now it indicates that one file has been changed. A message at the bottom says "create one data.txt file".

Task 4: Handling Runtime Errors Gracefully in Loops

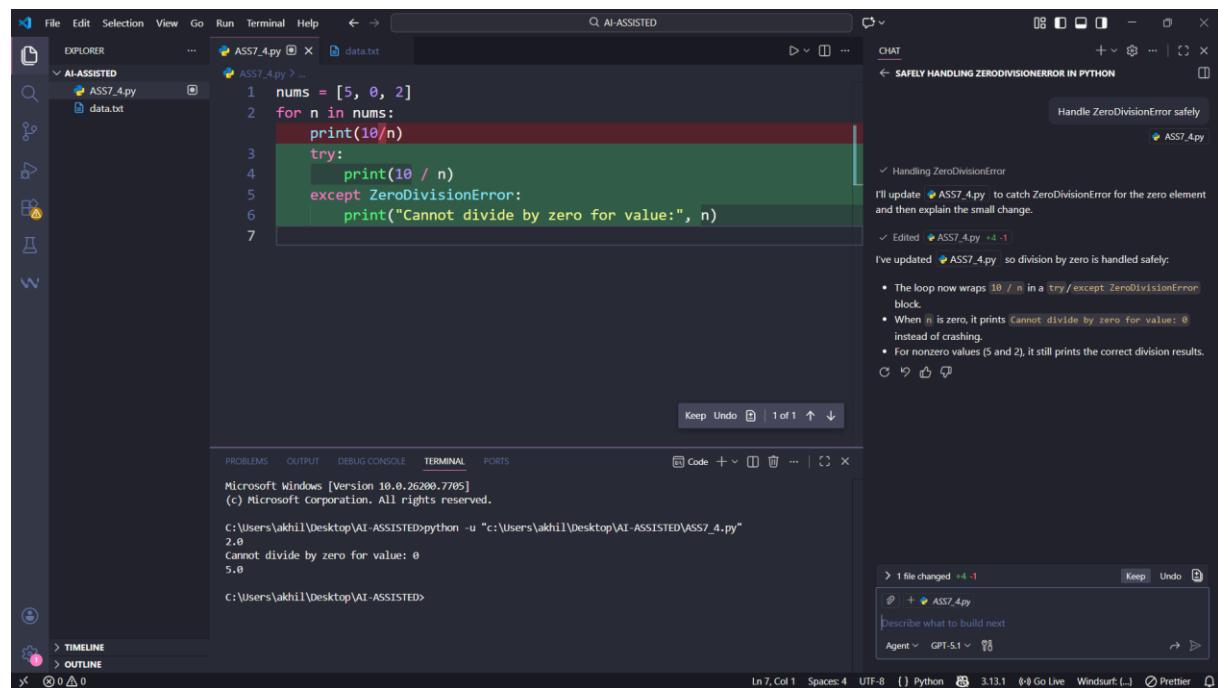
Scenario

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.



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar has 'EXPLORER' and 'AI-ASSISTED' sections. The main area shows a Python file 'ASS7_4.py' with the following code:`1 nums = [5, 0, 2]
2 for n in nums:
3 print(10/n)
4 try:
5 print(10 / n)
6 except ZeroDivisionError:
7 print("Cannot divide by zero for value:", n)`

To the right of the code editor is a 'CHAT' panel titled 'SAFELY HANDLING ZERO DIVISION ERROR IN PYTHON'. It contains a message from an AI assistant: 'Handle ZeroDivisionError safely' and 'ASS7_4.py'. Below this, a note says: 'I'll update ASS7_4.py to catch ZeroDivisionError for the zero element and then explain the small change.' A list of changes is shown:

- ✓ Handling ZeroDivisionError
- I've updated ASS7_4.py so division by zero is handled safely:
- The loop now wraps `10 / n` in a `try/except ZeroDivisionError` block.
- When `n` is zero, it prints `Cannot divide by zero for value: 0` instead of crashing.
- For nonzero values (5 and 2), it still prints the correct division results.

At the bottom of the interface, the terminal shows the output of running the script:`Microsoft Windows [Version 10.0.26200.7705]
(c) Microsoft Corporation. All rights reserved.

C:\Users\akhil\Desktop\AI-ASSISTED>python -u "c:\Users\akhil\Desktop\AI-ASSISTED\ASS7_4.py"
2.0
Cannot divide by zero for value: 0
5.0

C:\Users\akhil\Desktop\AI-ASSISTED>`

Below the terminal, status bar details include: Line 7, Col 1 | Spaces: 4 | UTF-8 | Python 3.13.1 | Go Live | Windsurf (...) | Prettier

The loop now wraps `10 / n` in a `try/except ZeroDivisionError` block.

When `n` is zero, it prints `Cannot divide by zero for value: 0` instead of crashing.

For nonzero values (5 and 2), it still prints the correct division results

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`)

Added `self` as the first parameter to `__init__`.

Assigned the parameters to instance variables `self.name` and `self.age`.