Coding & Development / Debugging

Take a stack trace and identify the root cause, suggesting where to add logging or breakpoints.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Stack Trace Analysis, Debugging, Root Cause
Updated: May 2026
Why This Prompt Exists
Most developers see a stack trace and start at the top — which is rarely the root cause.

You get:

- debugging from the wrong line (wasting time)
- missing the root cause buried in the trace
- no idea where to add logging or breakpoints
- ignoring the call chain that led to the error
- fixing symptoms instead of causes

But a stack trace is not random.

It is a map of the execution path.

- First frame: where the error was thrown (not always the cause)
- Your code frames: where your logic is (start here)
- Framework/library frames: rarely the problem
- Root cause: often several frames down

Without analysis, you fix the wrong line.

This framework forces AI to pinpoint the root cause.

The Prompt

```
Assume the role of a debugging expert who reads stack traces like a
detective reads clues.

Your task is to analyze a stack trace and identify the root cause.

Generate:

1. ROOT CAUSE IDENTIFICATION
   - Which line/file is the actual cause
   - Why (not where) the error occurred

2. CALL CHAIN EXPLANATION
   - How execution got to the error
   - Which frames are relevant vs. irrelevant

3. LOGGING SUGGESTIONS
   - Where to add console.log or logging statements
   - What to log at each point

4. BREAKPOINT RECOMMENDATIONS
   - Which lines to set breakpoints
   - What to inspect at each breakpoint

5. FIX SUGGESTION
   - Specific code change

INPUTS:
```

```
Stack Trace (paste full trace):
[PASTE STACK TRACE]

Language/Framework:
[PYTHON / JAVASCRIPT / JAVA / C# / OTHER]

Known Recent Changes:
[LIST OR "NONE"]

Error Type (if known):
[E.G., "Null reference," "Index out of range," "Type mismatch"]

RULES:
- Identify the first line of YOUR code in the stack (not library code)
- Explain the call chain (how execution reached the error)
- Ignore framework/library frames for root cause
- Suggest specific logging (not "add more logs")
- Suggest specific breakpoints (not "debug it")
- Don't assume you know the fix without enough context
```

How To Use It

- Paste the full stack trace (not just the error message).
- Identify which lines are your code vs. library code.
- The root cause is usually in your code, not the library.
- Follow the logging suggestions to instrument your code.
- Set breakpoints at the recommended lines and inspect variables.

Example Input

Stack Trace:

Traceback (most recent call last):
File “app.py”, line 45, in
result = process_data(user_input)
File “app.py”, line 32, in process_data
return [item.name for item in data_list]
File “app.py”, line 32, in return [item.name for item in data_list]
AttributeError: ‘NoneType’ object has no attribute ‘name’

Language/Framework: PYTHON

Known Recent Changes: Changed data_list to come from a new API endpoint instead of a local file

Error Type: AttributeError (‘NoneType’ object has no attribute ‘name’)

Why It Works
Most developers waste time on the wrong line.

This framework improves outcomes by forcing:

- root cause identification (accuracy)
- call chain explanation (understanding)
- specific logging suggestions (instrumentation)
- breakpoint recommendations (debugging)
- fix suggestion (resolution)

Great stack trace analysis doesn’t just read — it interprets and prescribes.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The Null/Undefined Error Fixer