Coding & Development / Debugging

Find and fix potential null or undefined reference errors with optional chaining, default values, or guard clauses.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Null Errors, Undefined Errors, Type Safety
Updated: May 2026
Why This Prompt Exists
Most runtime errors are null or undefined references — and they're entirely preventable.

You get:

- "Cannot read property 'X' of undefined" crashes
- errors that only appear in production (not dev)
- no defensive coding for missing data
- manual null checks everywhere (verbose)
- missed opportunities to use modern language features

But null errors are not inevitable.

They are preventable with defensive patterns.

- Optional chaining (?.) : safe property access
- Nullish coalescing (??) : default values for null/undefined
- Guard clauses: early returns for invalid state
- Type narrowing: checks that inform the type system
- Default parameters: fallback values for function arguments

Without prevention, null errors crash your app.

This framework forces AI to add defensive null/undefined handling.

The Prompt

```
Assume the role of a defensive programming expert who prevents null
and undefined errors.

Your task is to add null/undefined safety to code.

Generate:

1. PROBLEM LOCATIONS
   - Lines where null/undefined errors could occur

2. FOR EACH PROBLEM:
   - Why it could be null/undefined
   - Impact if it fails

3. FIX APPLIED (choose best approach)
   - Optional chaining (?.) for nested access
   - Nullish coalescing (??) for default values
   - Guard clause (early return)
   - Default parameter value
   - Type narrowing check

4. FIXED CODE (complete)

5. PREVENTION TIPS
   - How to avoid similar issues in the future
```

```
INPUTS:

Code (paste):
[PASTE CODE]

Language:
[JAVASCRIPT / TYPESCRIPT / PYTHON / OTHER]

Known Data Shape (describe expected structure):
[E.G., "user object with profile.name, optional settings.theme"]

Error History (if known):
[E.G., "Sometimes API returns null for user"]

Risk Level:
[LOW / MEDIUM / HIGH]

RULES:
- Use optional chaining (?.) for nested property access
- Use nullish coalescing (??) for default values (not ||)
- Use guard clauses (if (!data) return) at function start
- Use default parameters for function arguments
- Don't over-defend (trust your types when safe)
- Consider the difference between null and undefined
- Add comments explaining why the check is needed
```

How To Use It

- Use optional chaining (?.) for nested property access — it’s cleaner than multiple if checks.

- Use nullish coalescing (??) for default values — not || (which treats 0 and empty string as false).
- Use guard clauses at the beginning of functions — return early if data is missing.
- Don’t over-defend — if a property should never be null, let it crash (so you fix the root cause).
- Add TypeScript or Python type hints to catch these issues at compile time.

Example Input

Code:
function getUserName(user) {
return user.profile.name;
}

Language: JAVASCRIPT

Known Data Shape: user object with profile.name, but API sometimes returns null for user or missing profile

Error History: “Cannot read property ‘name’ of undefined” errors in production

Risk Level: HIGH

Why It Works
Most null errors are preventable.

This framework improves outcomes by forcing:

- problem location identification (precision)
- root cause analysis (why null/undefined)
- appropriate fix selection (context-aware)
- fixed code (actionable)

- prevention tips (learning)

Great null error prevention doesn’t just fix — it teaches you to write defensively.

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 Performance Bottleneck Finder