Prompts for Coding Help

Debug, explain and write code with AI — without copying answers you do not understand.

Introduction

AI tools are genuinely good at code — and genuinely dangerous for learners. They can explain an error in seconds, write a function you could not, and fix a bug you stared at for an hour. The danger is different: if you copy code you do not understand, you pass the assignment and fail the learning. This lesson teaches the three safe uses — **explain, debug, and draft** — and the one rule that keeps you honest: **never paste code you cannot explain back in your own words.**

(Roman Urdu: AI code mein madadgar hai, lekin khatra ye hai ke samjhe baghair copy kar lo. Teen mehfooz istemal hain: samjhao, bug dhoondo, draft banao — aur aik usool: jo code wapas apne alfaaz mein samjha na sako, wo paste mat karo.)

Key points

- **Use 1 — Explain:** paste code you wrote (or your teacher's example) and ask what each part does. This is the fastest way to learn real code.
- **Use 2 — Debug:** paste your broken code PLUS the error message PLUS what you expected. The error message is half the diagnosis — without it, the AI guesses.
- **Use 3 — Draft, then rewrite:** let the AI draft a small function, then rewrite it yourself in your own style, and test it. The rewriting is where the learning happens.
- **Always paste the full context:** the language, the code, the error, and what should happen. "My code is not working" with no code and no error gets you a generic lecture.
- **Ask for the explanation, not just the fix.** "Fix it and explain why it was broken in one sentence" teaches you; "fix it" alone teaches nothing.
- **Test every AI-written code yourself.** Run it. Change one value. Break it on purpose. AI code can look perfect and fail on edge cases.
- **Never submit AI code as your own learning.** For marked work, your teacher's rules decide what is allowed — when in doubt, ask. Using AI to understand is learning; submitting its output unread is not.
- **Keep secrets out of pasted code.** Passwords, API keys, personal data — never paste them into any AI chat. Replace them with placeholders like `YOUR_KEY_HERE` first.
- **Beginners get stuck on which error to paste.** Paste the FULL error message, not your summary of it ("it shows an error"). Error messages contain line numbers and exact words — "Cannot read properties of undefined" and "is not a function" point to different bugs, and your summary erases the difference.
- **Ask for a small test, not just a fix.** "After the fix, show me one test: what the code prints for input 5" proves the fix works. A fix with no test is a claim; a fix with a test is evidence.

Examples

Example 1 — Explain code you do not fully understand

```
Explain this JavaScript code to me line by line, in simple English, as if I am a beginner.
After each line, say what it does in one short sentence. Code:
let total = 0;
for (let i = 1; i <= 5; i++) {
  total += i;
}
console.log(total);
```

What you will get: a walk through the loop — total starts at 0, i counts 1 to 5, each value is added, the final total is 15. This is the best use of AI for a student: understanding, not copying.

Example 2 — Debug with the error included

```
My JavaScript code should print numbers 1 to 5, but it prints nothing. Here is the code:
for (let i = 1; i > 5; i++) {
  console.log(i);
}
What is wrong, which line is the problem, and what is the one-line fix? Explain the fix in one
sentence.
```

What you will get: the condition `i > 5` is false on the first check (1 is not greater than 5), so the loop never runs — fix it to `i <= 5`. Notice the prompt includes the code, the expected result, and the actual result. That is the debugging recipe: code + error + expectation.

Example 3 — Draft, then make it yours

```
Write a small JavaScript function called isPass that takes marks and returns true if marks are
40 or more, false otherwise. Keep it under 10 lines, simple style, with one example of calling
it.
```

Then, after you get it: rewrite it yourself without looking, run it with marks 35 and 65, and explain in one line why 40 is the boundary. If you can do that, the code is yours — the AI only drafted it.

Example 4 — Before and after: a debug prompt without and with the error

Before:

```
My code is not working. Fix it.
```

After:

```
My JavaScript code should print the even numbers from 1 to 10, but it prints nothing. Here is
the code:
for (let i = 2; i < 10; i = i + 2) {
  console.log(i);
}
There is no error message — it just prints nothing in my browser console. What is wrong, and
what is the one-line fix? Explain the fix in one sentence.
```

The after version carries code, expected result, actual result, and the error situation (none). Even "no error message" is useful information — it tells the AI the code runs but never enters the loop, which points at the loop condition, not a typo.

Example 5 — Draft, then make it yours: a marks average function

```
Write a small JavaScript function called average that takes an array of marks and returns
their average. Keep it under 10 lines, simple style. Show one example: average([80, 90, 70])
should give 80.
```

Then, without looking: rewrite it yourself, run it with [50, 100], and explain in one line what happens if the array is empty. If the empty-array answer stumps you, ask the AI: "What does my function do for an empty array, and how do I handle it?" That follow-up is the learning — edge cases are where copied code fails first.

Example 6 — Explaining code in two directions

```
Explain this code in two parts. Part 1: what each line does, in simple English, one sentence
per line. Part 2: what the whole program does in one sentence, as if explaining it to a
friend. Code:
let names = ["Ali", "Sara", "Ahmed"];
for (let name of names) {
  console.log("Hello " + name);
}
```

Line-by-line alone can leave you knowing the parts but not the purpose. The one-sentence summary forces the whole picture: "It greets each name in the list." Students who can do both — parts and purpose — are the ones who can change the code next week.

Real-world case — Where this is actually used

Junior developers at real companies use AI the same three ways: explain code they inherited, debug with the error message pasted, and draft a small function they then rewrite and test. The rule is identical in a classroom and an office — nobody pastes AI code into a real project without running it, and nobody pastes passwords or keys into a chat, ever. A student who practises "explain, debug, draft, test" now is practising the exact workflow working developers use daily.

Quick recap

- Three safe uses: explain, debug (code + error + expectation), draft-then-rewrite.
- Always include language, code, error message, and expected result.
- Ask for the explanation with the fix, not the fix alone.
- Test AI code yourself; never submit what you cannot explain.
- Never paste passwords, keys or personal data.

Common mistakes

1. **"My code is not working" with nothing pasted** — the AI cannot diagnose air. Fix: paste code + error + what should happen.
2. **Copying a fix without understanding it** — the same bug returns next week. Fix: ask "why was it broken, in one sentence" every time.
3. **Pasting secrets into the chat** — keys and passwords in a prompt can leak. Fix: replace them with placeholders before pasting, every time.

1. **Pasting only the broken line** — one line without the lines around it hides the real cause (a variable set wrong five lines above). Fix: paste the whole small function, not just the line the error points at.
2. **Accepting a fix that changes the whole program** — the AI rewrites 30 lines to fix a one-line bug, and you no longer recognise your own code. Fix: ask for "the smallest possible fix — change as few lines as you can, and tell me which lines you changed".

Practice

1. Take a working program from your course. Ask the AI to explain it line by line, then close the chat and explain it yourself to a friend (or out loud). Note which lines you still cannot explain — those are your real study targets.
2. Break a small program on purpose (change one operator). Debug it with the AI using the code + error + expectation recipe, and write down the one-sentence explanation of the bug.

1. Take an error message you do not understand. Ask the AI to explain each part of the message in simple words (which line, which word, what it means), without fixing the code yet. Then ask for the fix separately — notice how understanding the message first makes the fix make sense.
2. Ask the AI to draft a small function (for example: check if a number is even). Rewrite it yourself without looking, run it with 3 inputs including an edge case (0 or a negative number), and note any input where your version and the AI version differ.
3. Take a program from your course that you can run but cannot fully explain. Ask for the two-part explanation (line by line, then one-sentence purpose). Then close the chat and explain the purpose to a friend in your own words.

Practice quiz

1. What three things must a debugging prompt include?

- Your name, your class, your school
- The code, the error message, and what you expected to happen
- The answer you want, copied
- Only the error message

Answer: The code, the error message, and what you expected to happen

2. What is the rule that keeps AI coding help honest for a learner?

- Never use AI for code
- Never paste code you cannot explain back in your own words
- Always copy the longest answer
- Only use AI for homework

Answer: Never paste code you cannot explain back in your own words

3. What must you never paste into an AI chat?

- Your code
- Passwords, API keys, CNIC numbers and other private data — replace them with placeholders first
- Error messages
- Comments in your code

Answer: Passwords, API keys, CNIC numbers and other private data — replace them with placeholders first

---

Watch the video lesson: https://youtu.be/8iNVV8IA--I

Read online: https://scopetreasure.com/notes/pe-coding-help · More free tools and notes at https://scopetreasure.com