C02 # Prompting Basics for Claude Code

Conceptual Beginner

The principles that make the difference between a good and a bad prompt

Context > Length

A 3-line prompt with the right context produces better results than a vague 30-line prompt. Claude needs to know where it is in the project, what the precise objective is, and what it must not touch.

The question to ask before writing a prompt: “Would someone discovering this codebase today understand exactly what I want?” If the answer is no, the prompt lacks context.

Specificity: the word that changes everything

“Refactoring” says nothing. “Extract the `validateUser` function to `src/auth/validation.ts` while keeping all tests green and without modifying public types” says everything needed.

Vague phrasing	Precise phrasing
`Improve this code`	`Reduce cyclomatic complexity of parseConfig() without changing the public API`
`Add tests`	`Write unit tests for validateUser, cover null cases and malformed emails`
`Fix the bug`	`The sortItems function sorts in reverse order, fix it while keeping the same return type`

Short iteration: validate in increments

Asking for everything at once is tempting but risky. Claude can produce an internally consistent solution that does not match implicit expectations. The practice that works: decompose into steps of 20-40 lines maximum, validate each one before continuing.

```
# After each change, check before continuing
git diff
pnpm test
```

Show rather than explain

Giving a reference file is worth more than explaining the style in prose. If you want Claude to follow the conventions of `UserService`, simply indicate: “Follow the same pattern as `src/services/UserService.ts`.”

The principle applies to types as well: if you have an existing `ApiResponse<T>` type, show it instead of describing it.

What NOT to do

Rhetorical questions: “Can you refactor this function?” invites Claude to potentially answer “yes” then wait. Prefer the direct imperative: “Refactor this function.”

Implicit constraints: If tests must stay green, if public types must not change, if a specific library is forbidden — say it explicitly. Claude does not have access to your undocumented conventions.

Ambiguous scope: “This file” without a reference, “the authentication” without specifying which module. Use absolute paths or function names.

The efficient minimal prompt

```
Context: [file or module involved]
Objective: [precise action + success criterion]
Constraints: [what you must NOT modify]
```

Three lines are enough for the majority of common tasks.