

香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

CSC4801
AI-assisted Software Engineering

Lecture 05: How to Write a Prompt

Instructor: Jinsheng Ba | 2026 Fall

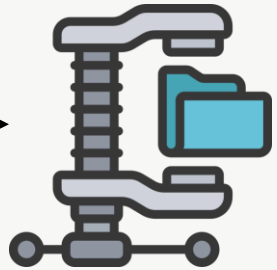


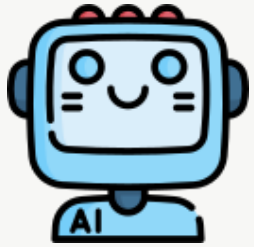
Skills



RAG

Prompts



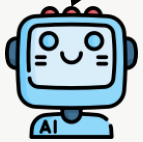


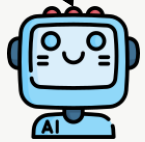
MCP



Sandbox

Context Management





Subagents

Prompt Is All You Need

All students in CSC 4801

Two Key Parts

SPECIFY

Goal and completion

Context and examples

VERIFY

Workflow and tools

Tests and iteration

Two Key Parts

Human intent

Task specification

Model-generated candidate

Review and execution

Observed behavior

Accept or revise

Prompt quality matters; acceptance still needs engineering evidence.

Keep Three Questions Visible

What should change?

What must stay true?

How will we know?

Write these questions before writing a long prompt.

A Prompt is a Conditional Input

Instructions describe

what to do and how to respond.

Context supplies

facts, examples, and evidence.

The model responds to the available context, not your unspoken intent.

Starting Point

GOAL

What should change?

CONTEXT

Which facts matter?

INPUT

What should be processed?

CONSTRAINTS

What must stay true?

OUTPUT

What should be delivered?

CHECKS

How will we know?

This is a classroom checklist, not a universal named framework.

Verb and Object

VAGUE

Help with my Python code.

Make it better.

SPECIFIC

Fix empty-list handling

in stats.average(xs).

A clear scope reduces accidental work and hidden choices.

Define Observable Behaviors

INPUT CONTRACT

xs is a list of finite numbers.

Empty lists are allowed.

OUTPUT CONTRACT

[] returns None.

Otherwise return the mean.

An edge case is part of the requirement, not a model preference.

```
def average(xs):
    return sum(xs) / len(xs)
```

Prompt: A Complete Request

```
Fix stats.average(xs) for empty lists.
Input: a list of finite numbers.
Return None for []; otherwise the mean.
Preserve the signature and input list.
Use only the Python standard library.
Return a minimal diff and test evidence.
```

Attach stats.py.

Include the three examples from the previous slide.

State if tests cannot run.

Short, concrete, and tied to a checkable outcome.

Let us build a game or ?

Role Prompts

ROLE ONLY

You are a senior engineer.

Review this code carefully.

ROLE + METHOD

Check input boundaries.

Find one concrete failure.

Cite the relevant line.

A role can guide style and perspective; it does not grant expertise.

Role Prompts

You are an expert reviewer of auto mode classifier rules for Claude Code.

Claude Code has an "auto mode" that uses an AI classifier to decide whether tool calls should be auto-approved or require user confirmation. Users can write custom rules in four categories:

- **allow**: Actions the classifier should auto-approve
- **soft_deny**: Destructive/irreversible actions the classifier should block unless clear user intent authorizes them
- **hard_deny**: Security-boundary actions the classifier should block unconditionally (user intent does not clear these)
- **environment**: Context about the user's setup that helps the classifier make decisions

Your job is to critique the user's custom rules for clarity, completeness, and potential issues. The classifier is an LLM that reads these rules as part of its system prompt.

For each rule, evaluate:

1. **Clarity**: Is the rule unambiguous? Could the classifier misinterpret it?
2. **Completeness**: Are there gaps or edge cases the rule doesn't cover?
3. **Conflicts**: Do any of the rules conflict with each other?
4. **Actionability**: Is the rule specific enough for the classifier to act on?

Be concise and constructive. Only comment on rules that could be improved. If all rules look good, say so.

https://github.com/Piebald-AI/claude-code-system-prompts/blob/main/system-prompts/agent-prompt-auto-mode-rule-reviewer.md

Separate Instructs and Data

TASK

Summarize the incident.

Use only the supplied log.

DATA

<incident_log>

...log text...

</incident_log>

Minimal Code Region

WHERE

Target path and function

Relevant caller or interface

Runtime / dependency version

WHY + HOW TO CHECK

Expected behavior

Minimal failing example

Relevant test command

Give enough context to distinguish the intended patch from a guess.

Example

Zero-shot Prompting

Classify the issue as:

BUG / FEATURE / QUESTION.

Return one label.

Issue: "How do I export CSV?"

Expected: QUESTION

Try a clear instruction before adding examples.

Zero-shot Prompting

- Extract the drink order from the message below. Output JSON.

- "A large iced latte, less sugar, with an extra shot of espresso"

Few-shot Prompting

"Crashes on startup" -> BUG

"Add CSV export" -> FEATURE

"Where is export?" -> QUESTION

Now classify:

"CSV export is missing rows"

Expected: BUG

Examples can communicate a decision boundary or local convention.

Few-shot Prompting

- Extract the drink order from the message. Follow the example format exactly; keep field names in English.

- Message: "hot americano, medium"
- Output: {"size": "medium", "drink": "americano", "temp": "hot", "sugar": "normal", "extras": []}

- Message: "large iced latte, no sugar, oat milk"
- Output: {"size": "large", "drink": "latte", "temp": "iced", "sugar": "none", "extras": ["oat milk"]}

- Message: "A large iced latte, less sugar, with an extra shot of espresso"
- Output:

Select Examples for Coverage

WEAK SET

Three obvious BUG reports

Same wording every time

No borderline case

STRONGER CANDIDATE SET

One example per label

An ambiguous-looking case

Consistent output format

Optimize coverage on your task, not an arbitrary example count.

Chain-of-Thought Prompting

Let’s think it step by step.

THE LIMIT

Results depend on model

and task setting.

More text is not proof.

Historical prompting results are evidence to test, not guarantees.

Planning Prompting

PLAN TOO VAGUE

1. Understand
2. Implement
3. Test

PLAN WITH DECISIONS

Read empty-input contract.

Add only an empty-list guard.

Run edge and regression tests.

A good plan names what each step will resolve.

https://platform.claude.com/docs/en/build-with-claude/structured-outputs

Structured Output Prompting

Review the following function.

Return JSON only.

```
{
  "bug": "",
  "severity": "",
  "reason": "",
  "corrected_code": ""
}
```

Structured output facilitates output processing.

A Template to Adapt, Not Blindly Copy

```
Goal: <observable change>
Context / input: <current material>
Constraints: <what must stay true>
Deliverable: <usable output>
Checks: <acceptance evidence>
If blocked: <report the missing fact>
```

OPTIONAL ADDITIONS

A relevant example

A short staged workflow

A source / version

A budget or stopping rule

Delete fields that add no value to your actual task.

The Habit Worth Keeping

Make intent explicit.

Bring the right evidence.

Check the artifact.

Revise from the failure.

A good prompt starts a reliable handoff; verification completes it.

What if the output is still incorrect after several attempts?


GAME OVER
MARIO
000000
x00
WORLD
A-1
TIME
WORLD A-1
x 3

Test-time Scaling

A Fixed Model Can Spend More Effort Per Task

TRAINING-TIME COMPUTE

Training data + optimization

Update model weights
Amortize cost across future tasks

TEST-TIME COMPUTE

One new task + extra inference

Keep model weights fixed
Spend extra effort on this task

Test-time scaling increases inference compute; it does not require updating model weights.

arxiv.org/abs/2408.03314

Three Ways

SEQUENTIAL

Reason longer

Revise after feedback

One path, more steps

PARALLEL

Sample candidates

Vote or score outputs

More paths to compare

SEARCH

Expand promising steps

Prune or backtrack

Choose where to explore

More depth, more candidates, and better search are different allocation choices.

arxiv.org/abs/2408.03314

Sequential Scaling

DRAFT

All customers
pay 80%

>

FEEDBACK

Nonmember:
expected 100, got 80

>

REVISE + RETEST

Only members
receive the discount

A later test must run on the revised code, not reuse an old result.

Extra steps help when they produce new evidence or correct a specific mistake.

arxiv.org/abs/2501.19393

Parallel Scaling

GENERATE -> SELECT

Self-consistency: vote on answers

Best-of-N: rank with a verifier

Both depend on selection quality

P(at least one correct) = 1 - (1 - p)^N

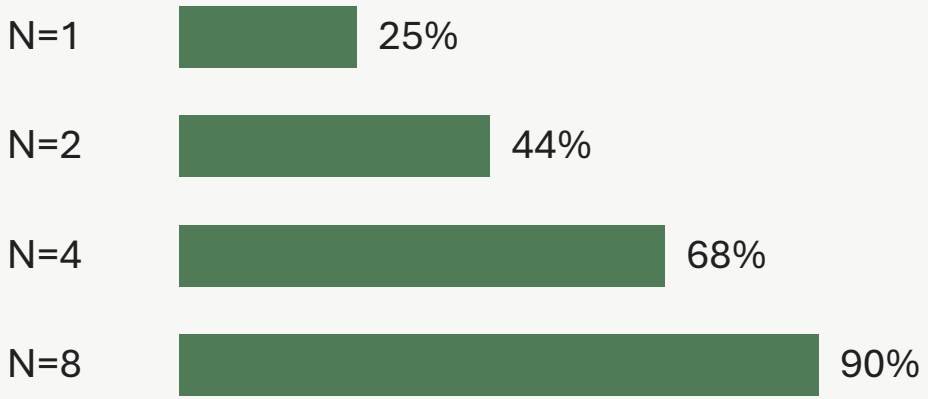
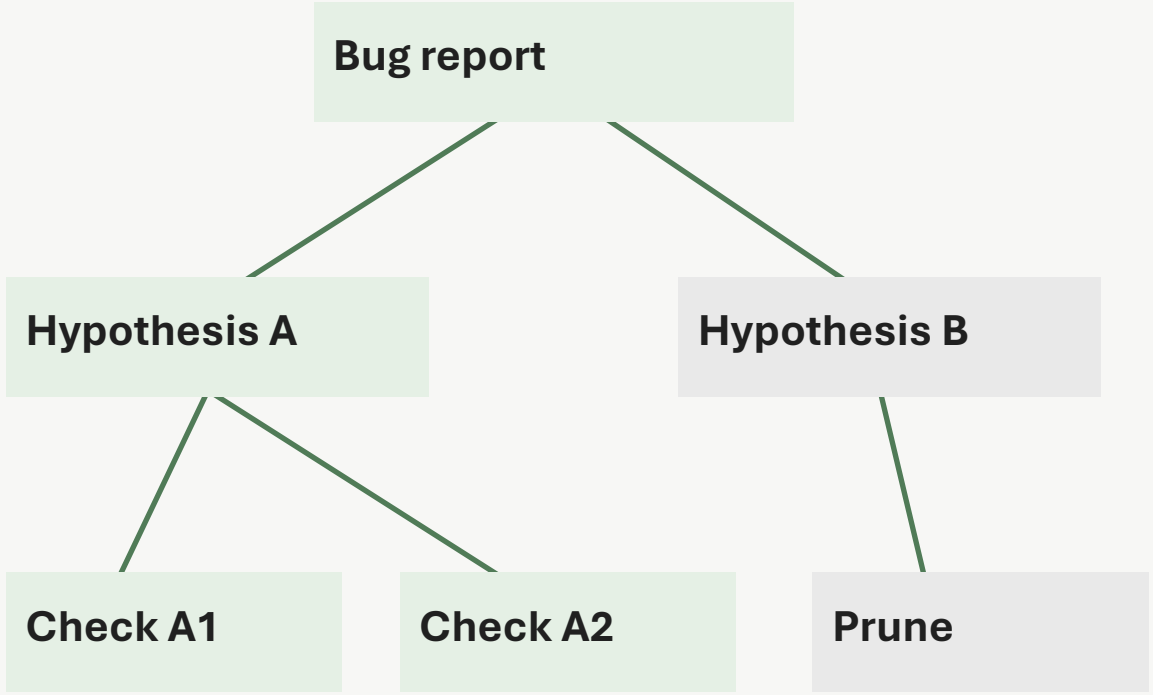


Illustration: p = 0.25; independent samples.

A correct candidate in the pool is useful only if the selector can identify it.

arxiv.org/abs/2203.11171

Search: Explore Branches



A SEARCH CONTROLLER

Generate a few branches
Evaluate intermediate states
Expand or backtrack
Stop when budget is exhausted

Search spends compute on promising intermediate states, not just complete answers.

arxiv.org/abs/2305.10601

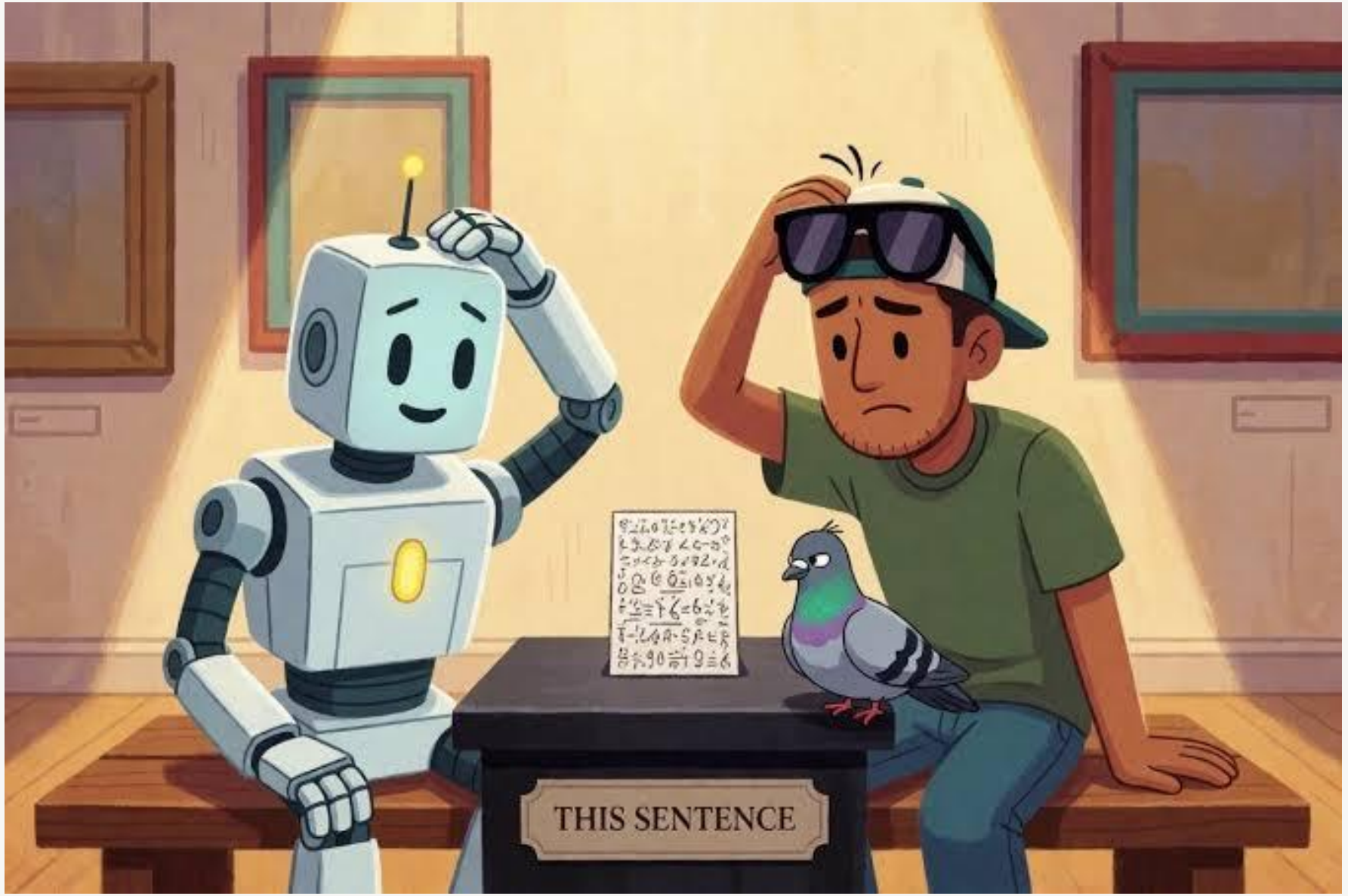
Allocate Effort by Difficulty

CLEAR + EASY	HARD + CHECKABLE	MISSING INFORMATION
One attempt	Try diverse candidates	Retrieve facts or ask
Run a cheap check	Use informative feedback	Change the approach
Stop if acceptance is met	Reserve budget to verify	Stop blind retries

The best use of extra compute depends on the task and the available verifier.

arxiv.org/abs/2408.03314

Still does not work?



You may need AGI



香港中文大學（深圳）
The Chinese University of Hong Kong, Shenzhen

Questions

Next class: Requirements