DAY FOUR

04 /10

Advanced Prompting & Context — *structure for hard work*

Day 3 covered the everyday brief. Today is for the heavier tasks: long instructions with many parts, large documents, persistent operating rules, and getting Claude to reason carefully before it answers.

BY THE END OF DAY 4, YOU WILL BE ABLE TO TEACH OTHERS TO —

- Structure long prompts with clear sections
- Use the persistent "operating instructions" layer
- Work with long documents and ground answers
- Separate instructions from source material
- Get Claude to reason before it answers
- Avoid the common long-context pitfalls

01 Why today matters

ORIENTATION

A short, clear brief carries everyday work beautifully. But real tasks are often not short: a prompt with five rules and three inputs, a forty-page document to analyse, a standing set of instructions that should apply every time. Without structure, complex prompts collapse into ambiguity — Claude can't tell the rule from the example from the data. Today is about giving complexity a clear shape.

Nothing here replaces Day 3 — Task, Context, Format and Examples still hold. Today simply adds the organising tools that keep those ingredients legible when there are a lot of them. Teach it as architecture, not as a new language.

PREREQUISITES

Day 3's four ingredients, and comfort iterating. This day is the bridge between the conversational surface (Days 1–2) and the developer surfaces (Days 5–8), where structure matters even more.

02 Give long prompts a clear shape

CORE CONCEPT

When a prompt has many parts, the failure mode is blur — Claude can't reliably tell your instructions from your data from your examples. The fix is visible structure.

2.1 Label the parts

Use headings or tags to mark each region of the prompt. Many people use XML-style tags because the open/close pairs are unambiguous and easy for the model to track:

```
<instructions>
   Summarise the report below for a non-technical executive.
   Keep it under 150 words. Flag any financial risk explicitly.
</instructions>

<report>
   [ … the full source document goes here … ]
</report>

<example_tone>
   [ … a past summary whose voice you want matched … ]
</example_tone>
```

The tags themselves aren't magic — plain headings work too. What matters is that each part is unmistakably its own thing.

2.2 Separate the instruction from the data

The most common structural bug: pasting a document and instructions together so Claude can't tell which is which — and treats your data as commands, or your commands as data. Always make the boundary explicit.

KEY IDEA TO INSTALL

Complex prompts need visible seams. Instructions, source material, examples and output spec should each sit in a clearly marked region. Structure is just kindness to the reader — and the reader here is Claude.

03 The persistent instruction layer

CORE CONCEPT

Some instructions shouldn't be retyped every message — they describe how Claude should operate for a whole body of work. Developers call this the system prompt. In the Claude apps you've already met its everyday forms: Project custom instructions, preferences, and styles.

> THE MENTAL MODEL
>
> **Standing orders vs. today's task.**
>
> The persistent layer is the standing orders — role, tone, rules, constraints, what to always or never do. The individual prompt is today's task. Keeping them separate means each prompt stays short, and the behaviour stays consistent across every conversation.

WHAT BELONGS IN THE PERSISTENT LAYER

- ✓ The role or perspective Claude should hold for this work
- ✓ Voice, tone and formatting conventions
- ✓ Hard rules and constraints — "only use sources I provide," "never give legal advice"
- ✓ Reference facts that every task in this context needs

WHAT DOES NOT BELONG THERE

The specifics of one task. If it changes per request, it's a prompt, not a standing order. Teach learners to ask: "Would I want this to apply to every conversation?" If yes, it's persistent; if no, it's per-prompt.

04 Make Claude reason before it answers

CORE CONCEPT

For anything logical, multi-step, or easy to get subtly wrong, the quality of the answer rises sharply when Claude works through it rather than leaping to a conclusion.

- **Ask for step-by-step reasoning.** "Work through this carefully before giving your answer." The visible reasoning is also easier for you to check.
- **Ask for a plan first.** For a big task: "Outline your approach before you start, and wait for me to confirm." This catches misunderstandings before effort is spent.
- **Use extended thinking for hard problems.** Claude can be given room to think more deeply before responding — well suited to genuinely difficult reasoning, where a fast answer is a risky answer.

> *"For hard problems, a fast answer is a risky answer. Ask for the reasoning."*

> TEACHING SHORTCUT
>
> Frame it as the difference between blurting and thinking. You wouldn't trust a colleague's instant answer to a tricky logic puzzle — you'd ask them to show their working. Same instinct, same phrase: "think it through first."

05 Working with long context

CORE CONCEPT

Modern Claude models can hold a great deal at once — long documents, large codebases, extended conversations. That power comes with a few habits worth teaching.

5.1 Ground the answer in the source

When you've supplied material, say so explicitly: "Answer only from the document above. If it isn't covered there, say so." Grounding pins Claude to your source and sharply reduces invented detail.

5.2 Put the question after the material

With a long document, a useful pattern is: source first, then instruction. It keeps the actual task clearly in view at the end rather than buried above a wall of text.

5.3 Mind the middle, and mind the size

Even with large context windows, detail in the middle of a very long input can get less attention than the start or end. For big jobs, teach learners to break the work down — summarise sections, then synthesise — rather than dumping everything and hoping. And remember Day 1: the window is large but finite.

> **THE LONG-CONTEXT TRAP**
>
> "I gave it everything, so it must have used everything." Not necessarily. Volume is not the same as attention. For high-stakes work over long material, ask for citations or quotes, and verify that the answer actually rests on the source.

LAB 04 ~30 MIN

Build a structured power-prompt

Each learner builds one well-structured prompt around a real document of their own and a multi-part instruction — then proves grounding works.

1. **Choose a real document** — a report, policy, long email thread, or article the learner actually needs to work with.
2. **Write a tagged prompt** with three labelled regions: an `<instructions>` block with at least three rules, the `<document>` itself, and an `<output_format>` spec.
3. **Add a grounding rule** — "answer only from the document; if something isn't covered, say so" — and run it.
4. **Test the grounding.** Ask a question the document genuinely doesn't answer. Confirm Claude says so rather than inventing.
5. **Add a reasoning step.** Re-run asking Claude to outline its approach first; compare the result to the leap-straight-in version.

a reusable structured prompt the learner can adapt for similar documents — and a first-hand demonstration that grounding stops invention.

TEACHING NOTES

How to teach Day 4 well

OPEN WITH THIS

Show a messy prompt where a document and instructions run together, and ask the room: “Which part is the rule and which part is the data?” When they hesitate, point out that Claude hesitates too. Structure is the fix they just asked for.

PACE & EMPHASIS

Sections 02 (structure) and 05 (long context) are the substance — anchor the day there. Section 03 connects straight back to Day 2’s Projects; lean on that link rather than re-teaching. Keep the tone “architecture,” never “syntax.”

DISCUSSION PROMPTS

· Where in your work do instructions and data currently run together? · What belongs in your persistent layer versus a single prompt? · What’s a task where you’d genuinely want to see Claude’s reasoning before trusting the answer?

COMMON MISCONCEPTIONS TO PRE-EMPT

“XML tags are a magic spell.”
They’re just unambiguous labels. Plain headings work too — clarity is the point, not the brackets.

“Bigger context means it read all of it carefully.”
Volume isn’t attention. Ground answers, ask for citations, break big jobs down.

“Persistent instructions are an advanced developer thing.”
They already used them on Day 2 — Project instructions, preferences, styles. Same idea.

IF YOU ONLY HAVE 30 MINUTES Teach visible structure (the tagged-prompt example), the grounding rule, and “ask for reasoning.” Run Lab steps 1–4. The persistent layer can be a callback to Day 2.

Day 4 Cheat Sheet

Structure	Mark each region of a long prompt — instructions, data, examples, output spec — with headings or tags.
XML-style tags	Open/close labels like <document>...</document> — unambiguous seams, not magic.
Instruction vs. data	Always make the boundary explicit so Claude doesn't confuse one for the other.
System prompt	The persistent operating-instruction layer; in the apps, Project instructions / preferences / styles.
Standing orders test	"Apply to every conversation?" Yes → persistent layer. No → per-prompt.
Reason first	Ask for step-by-step thinking or an upfront plan on hard, multi-step tasks.
Extended thinking	Give Claude room to think more deeply before answering genuinely difficult problems.
Grounding	"Answer only from the source; if it's not there, say so." Cuts invented detail.
Mind the middle	Detail mid-document gets less attention; break big jobs down and ask for citations.

Check for understanding

Five questions. Learners should be able to answer all five before Day 5.

1. What is the failure mode of an unstructured long prompt, and what's the fix?
2. Are XML-style tags required? What is actually doing the work?
3. Give two things that belong in the persistent instruction layer and one that doesn't.
4. Name two ways to get Claude to reason before answering, and the kind of task that calls for it.
5. What does "grounding" mean, and why is "I gave it everything" not the same as "it used everything"?

> **Answer notes —** 1) Blur — Claude can't tell instructions from data from examples; the fix is visible structure / labelled regions. 2) No; the unambiguous separation of parts is what matters — plain headings work. 3) Belongs: role, tone/format conventions, hard rules, always-needed reference facts; doesn't belong: the specifics of a single task. 4) "Think it through step by step" and "outline your plan first"; logical, multi-step, or error-prone tasks. 5) Telling Claude to answer only from the supplied source; volume isn't attention, so verify the answer rests on the material and ask for citations.

Day 4 in five lines

- Complex prompts need visible structure — labelled regions for instructions, data, examples and output.
- Always separate the instruction from the source material so Claude can't confuse them.
- The persistent instruction layer holds standing orders; the prompt holds today's task.
- For hard, multi-step work, ask Claude to reason or plan before it answers.
- Long context is powerful but not infinite attention — ground answers and verify against the source.

TOMORROW — DAY 5 → Meet Claude Code — agentic coding in your terminal