LAB EXERCISES

Maximizing Claude Code

5 labs · ~50 minutes total

19 May 2026

LAB 1

Bare prompt vs context-rich prompt

Time	10 minutes
Goal	Feel the quality gap between a bare prompt and a context-rich prompt — for the same task.

Steps

1. Open Claude Code in any folder (a real project if possible).
2. Run Prompt A (below). Save the output.
3. Run Prompt B (below). Save the output.
4. Compare style, correctness, and fit to an existing codebase.
5. Discuss with the person next to you: which one would you actually merge?

Expected outcome

- Prompt B fits the existing codebase (naming, error handling, no new deps).
- Prompt A is generic, possibly wrong, possibly ESM instead of CommonJS.
- You feel that 2 minutes of context = much less review time.
- Open question: where should this context live permanently so you don't retype?

Stretch goal

- Run Prompt B a second time, but ask for TypeScript output. Watch context override the default.

Reference prompts

```
# PROMPT A — bare

Write me a function to validate email.
```

```
# PROMPT B — context-rich

Project: BookShelf (Node.js + Express). CommonJS, not ES modules.
Existing validators live in utils/helpers.js as named exports
(addDays, format_date, calc_fine). Follow that pattern.
Error handling: return { ok: false, error: "..." }. Do NOT throw.
Accept: standard RFC 5322-ish.
Reject: empty, no @, no dot, spaces, > 254 chars.
Called from auth.js during register and login.
No new dependencies — pure JS only.
```

```
Task: add validate_email(email) to utils/helpers.js. Update exports.
Show me the diff only. Include 5 inline test calls at the bottom.
```

LAB 2

Install codebase-memory MCP

Time	5 minutes
Goal	Install and register the codebase-memory MCP server with Claude Code.

Steps

1. Open a terminal in a clean folder (not your project).
2. Clone the MCP repository: git clone https://github.com/deusdata/codebase-memory-mcp.git
3. Install dependencies: cd codebase-memory-mcp && npm install
4. Register the MCP with Claude Code (see prompt below).
5. Open claude in any folder, type /mcp and verify codebase-memory is listed.

Expected outcome

- Output of /mcp lists codebase-memory.
- No errors during install or registration.
- Server starts successfully when Claude Code uses it.

Stretch goal

- Also install one other MCP from the directory (filesystem, git, fetch). Practice the same pattern.

Reference prompts

```
# Register with Claude Code
claude mcp add codebase-memory \
  --command "node $(pwd)/server.js"

# Verify (inside claude prompt)
/mcp
```

LAB 3

Index the BookShelf legacy app

Time	10 minutes
Goal	Index the sample legacy app and ask grounded questions. Compare with non-indexed answers.

Steps

1. cd into the 04_Sample_Legacy_App folder.
2. Open Claude Code: claude
3. Run: /codebase-memory index — wait for it to finish.
4. Ask: "How does borrowing a book work in this app? Walk me through all files involved."
5. Note the answer quality and the files it references.
6. Disable the MCP (/mcp disable codebase-memory) and ask the same question again.
7. Compare the two answers. Re-enable the MCP when done.

Expected outcome

- With MCP: answer cites routes/loans.js, db.js, and the loans table schema.
- Without MCP: answer is more generic, may miss the cross-file flow.
- You see why grounding matters for legacy code.

Stretch goal

- Ask: "Where would I add an audit log? Suggest the least invasive change."
- Ask: "What functions touch the users table?"

LAB 4

Convert YOUR workflow into a Skill

Time	15 minutes
Goal	Take one task you do at least weekly, codify it as a SKILL.md, and register it with Claude Code.

Steps

1. Pick one recurring task. Examples: "write a PR description", "add a CRUD endpoint", "triage a bug ticket".
2. Open a notepad. Write down the steps you take when doing it manually. Aim for 5–10 steps.
3. Create the folder: .claude/skills/<your-skill-name>/ in your project.
4. Create SKILL.md inside it using the template below.
5. Restart Claude Code (or run /reload) and check /skills lists your new skill.
6. Trigger it by typing one of the trigger phrases. Watch it run.

Expected outcome

- Your skill appears in the /skills list.
- When you type one of the trigger phrases, the skill activates.
- Claude follows the steps in order without you re-explaining.

Stretch goal

- Add a Constraints section listing 2 things the skill should NEVER do.
- Add an Examples section with one Good Run and one Bad-Run-Corrected.
- Commit the .claude/ folder to git so your team can use it.

Reference prompts

```
# .claude/skills/<your-skill-name>/SKILL.md
---
name: <your-skill-name>
description: |
  <One sentence: what does it do, when do you use it?>
triggers: ["<phrase 1>", "<phrase 2>"]
---

# Steps
1. <First step>
2. <Second step>
...
```

```
# Constraints
- <Never do X>
- <Always check Y first>

# Examples
## Good run
<paste a real example of you doing this task well>
```

LAB 5

Use your Skill on a real bug

Time	10 minutes
Goal	Apply your new skill (or a pre-made one) to fix a real bug from BookShelf's BUG_TICKETS.md.

Steps

1. Open 04_Sample_Legacy_App/BUG_TICKETS.md.
2. Pick ONE ticket. Suggested first try: BUG-002 (loan never decrements available_copies).
3. Tell Claude Code: "Use <your skill name> to handle BUG-002."
4. Watch the skill run. Approve patches when it asks.
5. Run the seed + start scripts to verify the fix works.
6. Take a screenshot of the diff to share with the group.

Expected outcome

- BUG-002 fixed: borrowing a book now reduces available_copies.
- available_copies hits 0 → next borrow is rejected.
- Your skill, your steps, your output. Reusable next week.

Stretch goal

- Chain skills: use a code-reviewer skill on the patch you just made.
- Pick BUG-001 (SQL injection) and try the same flow — multi-file investigation.
- Write a regression test that locks in the fix.