

Homework — Lesson 08

3 exercises · No coding required for Exercise 1 · Estimated total time: 3–5 hours

How to use this homework

Work through the exercises in order. Each one builds on the previous. Exercise 1 is deliberate design thinking — no code needed. Exercise 2 is your first working AI script. Exercise 3 asks you to plan a real project of your own. All three have self-check criteria so you know when you're done.

Exercise 1 — Design your approach before you build

Estimated time: 30–45 minutes · **Tools needed:** Claude.ai (free tier), a text editor.

Before writing any code, good builders answer three design questions. In this exercise you'll pick a real problem you'd like to solve with AI, then work through those three questions in writing.

Part A — Pick a problem

Think of one real, specific task in your life or work that fits the pattern: *I get some text as input, and I'd like to get some useful text as output*. Examples:

- Converting rough meeting notes into a tidy action list.
- Sorting a pile of customer feedback emails into themes.
- Summarising long PDF reports you receive weekly.
- Drafting replies to common questions you get as a freelancer.

Write your chosen problem in one sentence: *"Given [input], produce [output]."*

Part B — Answer the three design questions

Write a short answer (2–5 sentences each) to these questions:

1. **Cloud or local?** Use the Decision Wizard on the Theory page to get a recommendation. Explain in your own words why that recommendation fits your problem.
2. **What's the core prompt?** Write a first draft of the prompt you'd use — using the Role + Context + Task + Format structure from Lesson 05. Keep it to one paragraph.
3. **How will you validate the output?** Describe how you'd check whether the AI's output is actually correct or useful. What would a bad output look like? What would a good one look like?

Part C — Test the prompt in chat

Go to **claude.ai** (or ChatGPT if you prefer). Paste the prompt from Part B. Provide a real or realistic sample of the input. Look at what comes back, then iterate at least once — make the output better with a follow-up instruction.

Self-check criteria for Exercise 1

- You have a one-sentence problem statement.
- You have written answers to all three design questions.
- You have run at least one prompt-and-one-iteration in a chat session.
- You could explain to a friend why you chose cloud vs local for this problem.

Exercise 2 — Build the Meeting Notes → Actions script

Estimated time: 60–90 minutes · **Tools needed:** Python 3, `anthropic` library, API key.

This is a guided build. Follow the steps below. By the end you'll have a working Python script that takes meeting notes as input and produces a structured JSON list of action items.

Step 1 — Set up

Make sure you have the `anthropic` library installed and your API key in an environment variable:

```
pip install anthropic
export ANTHROPIC_API_KEY="your-key-here"    # Mac / Linux
# Or: set ANTHROPIC_API_KEY=your-key-here  # Windows CMD
```

Step 2 — Write the script

Create a file called `meeting_actions.py`. Fill in the structure below. The prompt is partially written for you — your job is to finish it:

```
import anthropic
import json

MEETING_NOTES = """
[Paste 5-10 lines of real or made-up meeting notes here.
Include things like: who said they'd do what, by when.]
"""

PROMPT = f"""
You are a meticulous project manager.

Extract every action item from the following meeting notes.
Return ONLY a JSON array. Each item must have these fields:
    - "task": string, what needs to be done
    - "owner": string, who is responsible (use "Unassigned" if unclear)
    - "due": string, when it's due (use "Not specified" if unclear)

Do not include anything outside the JSON array.
Do not make up tasks that are not mentioned.

Meeting notes:
{MEETING_NOTES}
"""

client = anthropic.Anthropic()
```

Notice a few things about this script:

- It uses `claude-haiku-4-5` — the cheapest, fastest Claude model. For structured extraction tasks it works well.
- The prompt explicitly says "Return ONLY a JSON array" and "Do not include anything outside the JSON array." This reduces the chance of getting explanatory

prose mixed in.

- The `try / except` block handles the case where the model ignores the format instruction. Always add validation.

Step 3 — Run it and iterate

Run the script with your sample meeting notes. If the output is wrong or the format is off, adjust the prompt — without changing the code structure. What prompt change makes the most difference?

Step 4 — Add file input (optional extension)

Once the basic script works, try reading the meeting notes from a `.txt` file instead of having them hardcoded:

```
import sys

if len(sys.argv) > 1:
    with open(sys.argv[1], 'r') as f:
        MEETING_NOTES = f.read()
# then use MEETING_NOTES in the prompt as before
```

Now you can run: `python meeting_actions.py my_notes.txt`

Self-check criteria for Exercise 2

- The script runs without errors on your sample input.
- The output is a valid JSON list that parses correctly.
- Each action item has task, owner, and due fields.
- You iterated on the prompt at least once after seeing the first output.
- Your API key is in an environment variable — not written directly in the script.

Exercise 3 — Plan your own project

Estimated time: 45–60 minutes · **Tools needed:** A text editor (or paper and pen).

Now it's your turn to plan something real. This exercise is about thinking clearly, not about building yet — although if you finish early, start building.

Write a one-page project brief

Using what you've learned in this lesson, write a short document that covers:

1. **The problem (1–2 sentences).** What does this tool do? For whom? What does it replace or improve?
2. **Cloud or local? Why?** One paragraph. Reference the relevant factors (privacy, volume, hardware, budget).
3. **Does it need RAG?** Will the model need to answer from specific documents you own? If yes, describe those documents and how you'd load them. If no, explain why the model's existing training is enough.
4. **The three layers.** Describe each layer:
 - Interface: how will users give input? (a file, a chat UI, a form, a command)
 - Your code: what does it do between input and model call? (clean the input, build the prompt, parse the output)
 - The model: which model or service will you use, and why?
5. **The golden rules check.** For each of the four golden rules, write one sentence explaining how your project will comply:
 - Human in the loop: where and how?
 - Validation: what will you check and how?
 - Key and data protection: what sensitive data is involved, and how will you protect it?
 - Start small: what is the absolute minimum version you could build this weekend?
6. **First step (make it tiny).** What is the first concrete thing you'll do? Ideally: "Open a chat window, paste this prompt, and test it with this sample input."

Watch for scope creep in your own brief

If your "minimum version" section takes more than one paragraph to describe, the scope is too wide. Strip it back until the minimum version could be

demonstrated to a sceptical friend in five minutes. You can add complexity once the minimum works.

Self-check criteria for Exercise 3

- You have a one-sentence problem statement that is specific enough to build to.
- Your cloud-vs-local decision is justified, not arbitrary.
- You've considered whether RAG is needed and have a clear answer.
- All three layers are described — even if one layer is "there is no interface yet."
- You have a single, concrete first step that you could start in the next 30 minutes.
- The minimum version fits in a paragraph.

What's next

Lesson 09 covers safety, privacy, cost, and limitations in depth — the things that matter most as soon as your AI project touches real users or real data. Everything you planned in Exercise 3 will be relevant there.