

INSTRUCTOR GUIDE AND NOTES

Prompt engineering

Writing clear instructions for language models. A 50-minute lesson on the six parts of a prompt, techniques that work, testing, and staying safe with untrusted text.

Series	Hack Your Future, PrivacySafe Foundation's series for building what's next
Audience	Anyone who uses AI chat tools or APIs. No programming needed for the core lesson.
Length	50 minutes, with a 30-minute cut
Format	4 slides, live workbench demo, 4 slides, pair lab, discussion
You will need	Slides (light for print, dark for projecting), the prompt workbench page, flash cards, access to any AI chat tool for the lab
Facts checked	October 27, 2026. Tools change quickly. Re-check the links before you teach.

PrivacySafe Foundation, Inc. | Hack Your Future | These materials are dedicated to the public domain under CC0 1.0 and shared for the public good. The PrivacySafe and PrivacySafe Foundation names and logos are trademarks and are not included in that dedication. Product names in the references belong to their owners.

The lesson in five sentences

1. A model sees only the text in its context window: a system message, your prompt, and any data you include. It has no access to your files or your intentions unless you provide them^[1].
2. A good prompt has six parts: task, context, role or audience, constraints, output format, and examples^[2]^[1].
3. Techniques such as few-shot examples, step-by-step reasoning, structured output and prompt chaining help in specific situations, and each has costs^[3]^[4]^[5]^[12].
4. Test prompts on a small set of real and tricky inputs, change one thing at a time, and re-test when the model changes^[11].
5. A prompt is not a security boundary. Treat any text the model reads as untrusted, limit what the model can do, and keep secrets out^[8].

LEARNING OUTCOMES

By the end, students can:

1. Describe what a model sees and why wording and sampling change the answer.
2. Name the six parts of a prompt and find which are missing from a real prompt.
3. Rewrite a weak prompt into a clear, testable one.
4. Build a small test set and use it to compare two versions of a prompt.
5. Explain prompt injection and apply layered mitigations^[8].

Session plan

SEGMENT	MINUTES	NOTES
Before the demo, slides 1 to 4	12	Includes the prediction: score your own prompt. Collect written prompts silently.
Workbench demo	12	Sections 1 to 4. Section 5 is optional.
After the demo, slides 1 to 4	10	Techniques, iteration, safety, checklist.
Pair lab	10	Write a template, test on three inputs, swap and attack.
Discussion	6	Two questions.

30-MINUTE CUT

Skip the pair lab. Run sections 1 and 4 of the workbench only. Cover slides 1 and 3 after the demo. Use one discussion question.

Before class

- ☐ Optional: the lesson page has a walkthrough video of under four minutes, in a light version for the page and a dark version for projecting, each with a subtitles file. Play it as a recap, or share it with learners who missed the session. Check that it plays on the room computer first.
- ☐ Open the workbench page and try each section. Set browser zoom to about 125% for the back row.
- ☐ Give learners the prediction task in advance if you can: bring one real prompt, with **no private details**.
- ☐ Decide which AI tool the room will use for the lab, and confirm everyone can reach it. A local model works too, and avoids sending anything to a hosted service.
- ☐ Optional: to run section 5, start a local model server (for example Ollama) and allow this page's address with the `OLLAMA_ORIGINS` environment variable^[13].
- ☐ Print the flash cards, double-sided, flipped on the long edge. Cut along the dashed lines.
- ☐ Remind the room before the lab: do not paste passwords, keys or personal data into any tool.

WHAT THE WORKBENCH IS, AND IS NOT

The checker uses simple patterns. For a pasted prompt it guesses the six parts from keywords such as "you are", numbers, "return" and "for example". For a built prompt it reads the fields exactly. It flags vague words, length without a number, conflicting instructions, long text without delimiters, and a few sensitive-data shapes. Token counts are characters divided by 4. It cannot judge whether an answer is correct. Section 5 sends a prompt only to a model on the learner's own computer, and results vary by model and by run.

Slide notes: before the demo

Slide 1: Title

about 2 min

SAY

The same model can give a brilliant answer or a useless one depending on how you ask. Our promise: a repeatable method beats trial and error. Our job: build a prompt from parts, test it, and make it safe to reuse.

DO

Point at the six-segment bar. It returns on slide 3 as the six parts.

CHECK FOR UNDERSTANDING

Ask: "Think of a time an AI tool gave you a poor answer. What did you type?"

Slide 2: What a model actually sees

about 4 min

SAY

- The model sees text: a system message from whoever built the app, your prompt, and the data you include^[1].
- It predicts the next token, one at a time. It has no memory or files unless you put them in the context window.
- Sampling adds randomness, controlled by temperature, so the same prompt can give different answers^[2].
- Everything is just text, including instructions hidden inside a document^[8].

DO

Draw the stack on the board: system message, prompt, data, then the model.

CHECK FOR UNDERSTANDING

Ask: "Where would a customer's pasted email fit in this picture?" Answer: as input data, which the model treats as text like everything else.

WATCH FOR

Learners often think the model "knows" their company or earlier chats. It knows only what is in the current context, plus what it learned in training.

Slide 3: Anatomy of a good prompt

about 4 min

SAY

PART	WHAT TO DO
Task	One clear job, starting with a verb.
Context	What the model cannot know: purpose, background, definitions.

Role or audience	Who answers, or who the answer is for.
Constraints	Measurable limits and what to skip.
Output format	Bullets, a table, JSON with named keys, one label ^[2] .
Examples	One input and the output you want ^[2] ^[3] .

Walk the weak and strong example side by side. The strong version is not longer for its own sake: each added line removes a guess.

DO

Put the key instruction near the start or the end. Models can use details at the edges of a long prompt better than details buried in the middle^[7].

CHECK FOR UNDERSTANDING

Ask: "Which part would most change the weak prompt if we added it?" There is no single right answer. It depends on the task.

WATCH FOR

Do not teach magic phrases or emotional pressure as tricks. Their effects are unreliable and model-specific. Teach clarity.

Slide 4: What to watch for, and the prediction

about 2 min

SAY

Techniques: be specific and measurable, separate instructions from data, show one example, name the format, ask for steps on hard problems, change one thing at a time. Risks: vague or conflicting wording, injection, confident wrong answers, secrets in hosted tools, results that change run to run.

DO

Read the prediction aloud: "How many of the six parts does your last real prompt contain?" Give two minutes of silence for learners to write the prompt down and guess a number. No private details.

WATCH FOR

Learners may guess high. Most everyday prompts contain a task and little else.

Workbench demo script (about 12 minutes)

Section 1: Score a prompt you already use

4 min

SAY

The page opens with a weak example: "Write something good about our product launch. Make it short but detailed. Don't use jargon. Don't be boring. Include the key stuff." The checker finds a task and little else, flags vague words, length without a number, a short-versus-detailed conflict, and mostly negative instructions.

DO

Read the findings aloud. Click "Load a strong example": it shows six of six parts and no common issues. Then invite two or three learners to paste their own prompt (no private details) and read their count against their prediction.

CHECK FOR UNDERSTANDING

Ask: "Which finding would you fix first, and how?"

WATCH FOR

The pasted-prompt check guesses parts from keywords. If it is wrong about a learner's prompt, say so: the point is the conversation, not the count.

Section 2: Build a prompt from parts

3 min

SAY

Pick "Classify support tickets". The fields fill in and the assembled prompt appears. Each field maps to a part. Turn on "Wrap the input in tags" and "Ask for step-by-step reasoning" and watch the assembled prompt change.

DO

Edit the constraints and the examples. Show the checker switching to exact part detection when the built prompt is selected.

CHECK FOR UNDERSTANDING

Ask: "What happens to the output if we remove the examples?" Expect: the labels may drift. Link to [few-shot prompting^{\[3\]\[2\]}](#).

Section 3: Before and after

2 min

SAY

Three pairs: summarize, extract, and plan a talk. For each, read the weak prompt, then the strong one, then the list of what changed.

DO

Use the buttons to load the weak and strong versions into the checker and compare the counts.

WATCH FOR

Strong prompts are not magic. They remove guesses, which makes results easier to test.

Section 4: Injection lab

3 min

SAY

A customer email contains a hidden instruction, highlighted. With no defenses, the model sees it as more text, and many models will follow it. Switch on delimiters, the untrusted-data statement and a fixed output format, and the prompt changes. The exposure meter is a teaching rating, not a measurement.

DO

Toggle each defense and read the assembled prompt. Then read the "Outside the prompt" list: least privilege, validating output, human review^[8].

CHECK FOR UNDERSTANDING

Ask: "If all three defenses are on, is the system safe?" No. Layered defenses help, but a prompt cannot guarantee the model ignores injected text^[8].

WATCH FOR

Direct injection comes from the user. Indirect injection hides in content the model reads, such as a web page, file or email^{[8][10]}. The lab shows the indirect case.

Section 5 (optional): try it on your own local model

as time allows

SAY

Choose the injection lab prompt and send it to a local model. Run it with defenses off, then on, at low temperature. Run the same prompt twice. The answers may differ between runs and between models.

DO

Requires a local model server and the origin allowed through `OLLAMA_ORIGINS` ^[13]. The page only talks to localhost.

WATCH FOR

If the request is blocked, the page shows the likely causes. Some browsers ask permission to reach the local network or block it from secure pages. Have a screenshot ready as a fallback.

Slide notes: after the demo

Slide 1: Techniques that work

about 3 min

SAY

TECHNIQUE	TEACHING POINT
Zero-shot	Instructions only. Fine for simple, common tasks ^[2] .
Few-shot	Examples show the pattern. Google's guidance recommends including them ^[2] , building on in-context learning research ^[3] . Keep examples varied so they do not bias the answer.
Step by step	Asking for or showing reasoning helps multi-step problems in capable models ^{[4][5]} . Models trained to reason may already do this, so check the model's guidance.
Structured output	Name a format and keys, then validate the result before using it ^[2] .
Prompt chaining	Split a big job into small steps, each with its own prompt ^[12] .
Self-consistency	Sample several reasoning paths and take the most common answer ^[6] . It costs several runs.

CHECK FOR UNDERSTANDING

Ask: "Which of these would you try first on the invoice extraction task, and why?" Few-shot with structured output.

WATCH FOR

Technique results come from specific models and tasks. Always test on yours.

Slide 2: Iterate and evaluate

about 3 min

SAY

The loop: write, test on varied inputs, compare results, change one thing, record it. Build a small test set of 5 to 10 real inputs with edge cases and what a good answer looks like. Use templates with variables. Log the version, the change and the result. Lower the temperature for consistency. Re-test when you switch models, because formatting and wording can change results^[11].

CHECK FOR UNDERSTANDING

Ask: "Why change only one thing at a time?" Otherwise you cannot tell which change helped.

Slide 3: Prompts and safety

about 4 min

SAY

RISK	TEACHING POINT
------	----------------

Prompt injection	Text in the data overrides your instructions ^{[8][9]} . Separate instructions from data, limit what the model can do, validate output, keep a human in the loop.
Hallucination	A confident answer that is wrong. Ask for answers drawn from text you supply and verify important claims ^[14] .
Sensitive data	Do not paste secrets or personal data into hosted tools. Redact, or use a local model where it fits.
Prompt leakage	System prompts can be revealed, so never put secrets in them ^[8] .

CHECK FOR UNDERSTANDING

Ask: "What is one thing you would not paste into a hosted AI tool?"

WATCH FOR

Do not leave the room thinking delimiters solve injection. They help. They do not guarantee.

Slide 4: Writing and running any prompt

about 4 min

SAY

1. Is the task one clear job?
2. Did I give context the model cannot know?
3. Did I say who the answer is for?
4. Did I set measurable limits for length, format and tone?
5. Did I show an example of what good looks like?
6. Did I separate instructions from the data?
7. Did I test on varied and tricky inputs?
8. Did I check the facts and keep secrets out?

DO

Introduce the pair lab below.

Pair lab (about 10 minutes)

1. Pick a task you repeat, such as summarizing meeting notes, answering a common email, or extracting dates from text.
2. Write a **template** with the six parts and a clearly marked place for the changing data. Use the template on the slide as a start.
3. Score it in the workbench. Fix the top finding.
4. Run it on **three inputs** in your AI tool: a typical one, a messy one, and one that tries to hijack it (for example, text that says to ignore your instructions).
5. Swap templates with another pair. Try to break theirs. Report what broke and how they could fix it.

PEER-REVIEW RUBRIC

- Is the task one job?
- Are the constraints measurable?
- Is the format checkable by a person or a program?
- Is the data fenced and labeled as data?
- Did it survive the hijack attempt, and what else would protect it?

Discussion (about 6 minutes)

QUESTION 1: FROM HABIT TO TEMPLATE

"Pick a task you repeat. Turn your prompt into a template: which parts change each time, and which stay fixed?"

- Fixed: role, constraints, format, examples. Changing: the input data and sometimes the context.
- Templates make prompts testable and shareable. Version them like code.

QUESTION 2: THE PASSWORD

"If a prompt says 'never reveal the password', what else must protect the password?"

- Do not put the password in the prompt at all. Prompts can leak^[8].
- Keep secrets in systems the model cannot read, with access checks outside the model.
- Least privilege, output validation and human review^[8].

Answer key and assessment

QUESTION	ANSWER
Name the six parts of a prompt.	Task, context, role or audience, constraints, output format, examples.
Why can one prompt give different answers?	Sampling randomness, controlled by temperature, and differences between models and versions ^[2] .
What does a model see?	Its context window: system message, prompt and data. No files or memory unless supplied ^[1] .
Few-shot versus zero-shot?	Few-shot includes examples. Zero-shot gives instructions only ^{[3][2]} .
Is a prompt a security boundary?	No. Use least privilege, validate output, and keep a human in the loop ^[8] .
Where should key details go in a long prompt?	Near the start or the end ^[7] .

EXIT TICKET (3 MINUTES)

1. Rewrite "Write something good about our product launch" with at least three of the six parts.
2. Name one reason to wrap pasted text in tags.
3. Name one thing you will not paste into a hosted AI tool.

HOMEWORK

- Choose one prompt you use weekly. Write it as a template, build a five-input test set including one tricky case, and report what you changed after testing.
- Compare two versions of a prompt on the same inputs and describe which changed the result most.
- Stretch: build a two-step prompt chain for a task that is too big for one prompt.

Common questions and misconceptions

QUESTION OR BELIEF	RESPONSE
"There is a secret phrase that always works."	No. Effects of phrases vary by model and task. Teach clarity and testing ^[11] .
"Longer prompts are better."	Only if each line removes a guess. Extra text can bury the key instruction ^[7] .
"Step by step always helps."	It helps many multi-step tasks ^{[4][5]} , but models trained to reason may do it already, and it costs tokens and time.
"Delimiters prevent injection."	They help the model and slow simple attacks. They do not guarantee safety ^[8] .
"If it worked once, it works."	Sampling varies. Test on several inputs and re-run ^[2] .
"Local models are always private and safe."	Local keeps data on your machine, but injection and wrong answers still apply.

References and further reading

Citations in this guide use bracketed numbers. Links were checked on October 27, 2026. Research papers are cited for what they showed on the models they tested. Confirm current vendor guidance before you teach, because it changes quickly.

#	SOURCE	WHAT IT SUPPORTS
[1]	OpenAI API documentation, Prompt engineering https://developers.openai.com/api/docs/guides/prompt-engineering	Message roles with different levels of authority; role and tone in the system message, task details and examples in user messages; few-shot structure.
[2]	Google, Prompt design strategies (Gemini API) https://ai.google.dev/gemini-api/docs/prompting-strategies	Clear instructions, constraints, response format, and the recommendation to include few-shot examples.
[3]	Brown et al., Language Models are Few-Shot Learners (2020) https://arxiv.org/abs/2005.14165	In-context learning from a few examples in the prompt.
[4]	Wei et al., Chain-of-Thought Prompting Elicits Reasoning in Large Language Models (2022) https://arxiv.org/abs/2201.11903	Examples that include intermediate reasoning steps improve multi-step reasoning in sufficiently large models.
[5]	Kojima et al., Large Language Models are Zero-Shot Reasoners (2022) https://arxiv.org/abs/2205.11916	A trigger phrase such as "Let's think step by step" can elicit reasoning without examples.
[6]	Wang et al., Self-Consistency Improves Chain of Thought Reasoning (2022) https://arxiv.org/abs/2203.11171	Sampling several reasoning paths and taking the most consistent answer.
[7]	Liu et al., Lost in the Middle: How Language Models Use Long Contexts (2023) https://arxiv.org/abs/2307.03172	Models can use information near the start and end of a long context better than information in the middle.
[8]	OWASP GenAI Security Project, LLM01:2025 Prompt Injection https://genai.owasp.org/llmrisk/llm01-prompt-injection/	Definitions of direct and indirect prompt injection, example attacks, and mitigations such as least privilege and human review.
[9]	Perez and Ribeiro, Ignore Previous Prompt: Attack Techniques for Language Models (2022) https://arxiv.org/abs/2211.09527	Early research on goal hijacking and prompt leaking attacks.
[10]	Greshake et al., Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection (2023) https://arxiv.org/abs/2302.12173	Indirect prompt injection through content a model retrieves or reads.
[11]	Sclar et al., Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design (2023) https://arxiv.org/abs/2310.11324	Small changes in prompt formatting can change results substantially, so re-test when wording or models change.

[12]	DAIR.AI, Prompt Engineering Guide https://www.promptingguide.ai/	A free, community-maintained overview of techniques such as prompt chaining and templates.
[13]	Ollama documentation, FAQ https://docs.ollama.com/faq	Running a local server on 127.0.0.1 and allowing extra web origins with OLLAMA_ORIGINS.
[14]	NIST, AI Risk Management Framework https://www.nist.gov/itl/ai-risk-management-framework	A framework for managing AI risks, including those of generative AI.
[15]	Creative Commons, CC0 1.0 Universal https://creativecommons.org/publicdomain/zero/1.0/	The public-domain dedication that covers these lesson materials.

ABOUT THESE MATERIALS

The slides, workbench, flash cards and this guide were written for the PrivacySafe Foundation Hack Your Future series. They are dedicated to the public domain under CC0 1.0^[15]. The Foundation names and logos are trademarks and are not included in that dedication. This lesson is independent of any AI provider named in the references.