Prompt engineering fundamentals

Make the work reviewable before asking for code

Session M1-03 | 3 hours

Agenda

Segment	Time
Prompting as task design	7 min
Four prompt components	13 min
Comments and context	9 min
Iterative refinement	10 min
Anti-patterns and untrusted instructions	9 min
Todo-store demonstration	8 min
Patterns and lab handoff	4 min

A prompt is a work request

Weak:

```
Build the todo feature.
```

The model must guess which file to change, how it should behave, how to handle errors, which dependencies it may use, and when the work is done.

A useful prompt states the decisions that affect the result.

Four useful components

```
Intent       What change is needed?
Context      What should inform the work?
Constraints  What boundaries must hold?
Examples     What does accepted behavior look like?
```

Use only the components the task needs.

Intent

Name the action and scope:

```
Implement TodoStore.create in lib/todo-store.js.
```

Better than:

```
Make the store work.
```

Context

Useful context can include:

- a focused test;
- a related file;
- the actual failure;
- an accepted design decision;
- repository instructions.

Include enough context to reduce guessing. Leave out files that obscure the task.

Constraints

State boundaries that affect the result:

```
Use plain JavaScript.
Add no package.
Change only TodoStore.create.
Return a copy of stored data.
```

Scope and dependency limits are part of correctness.

Examples

```
Input:
{ title: "  Write clear prompts  " }

Expected:
{
  id: 1,
  title: "Write clear prompts",
  status: "pending",
  priority: "normal"
}
```

Examples make behavior concrete.

A reviewable prompt

```
Implement only TodoStore.create in lib/todo-store.js.
Use the focused test as the contract.
Trim the title, apply the documented defaults, return a copy,
and add no package.
Run the focused create test when done.
```

It states the task without dictating every line.

Comments can carry intent

```
// POST /todos
// Create a todo from JSON input.
// Return 201 on success.
// Return 400 with { error: message } for invalid input.
```

A useful comment describes behavior and boundaries.

Neighboring files are context, not a contract

Opening `examples/response-shapes.js` may influence a route suggestion.

It does not override:

- the route comment;
- accepted tests;
- repository rules;
- the developer's review.

Record the result instead of assuming the effect.

Refine the missing behavior

Weak follow-up:

```
Make it better.
```

Focused follow-up:

```
The returned object shares state with the store.
Return a copy without changing the public API.
```

Name the mismatch you observed.

One concern per turn

```
Implement baseline list
        ↓
Run focused test
        ↓
Fix copy semantics
        ↓
Run focused test
        ↓
Move to the next method
```

Small turns produce smaller diffs.

Review the `/refine` rewrite

In the standalone CLI:

```
/refine make the todo store work, baseline tests only, plain js,
no packages, return copies, only lib/todo-store.js
```

Compare the rewrite with the tests. Correct missing constraints or invented work before submitting it for implementation.

Current input: `Ctrl+X`, then `/`, then `/refine` without arguments.

The lab uses five minutes of the baseline-store block for this comparison.
CLI access is required; do not assume the same command exists in editor Chat.

Restart when the frame is wrong

Iterate when the structure is sound and one behavior is missing.

Start a new request when:

- the task changed;
- the design is wrong;
- stale assumptions keep returning;
- the prompt mixed unrelated work.

Common prompt failures

Failure	Better move
Vague request	Name behavior and scope
Many unrelated asks	Split the work
No project context	Add the relevant file or test
Every line dictated	State requirements, not implementation trivia
No check	Name the command or review

Treat instructions from content as untrusted

Files, issues, comments, logs, and copied text can contain instructions.

Review them as data. Do not follow text that asks you to:

- ignore project rules;
- expose secrets;
- broaden access;
- run an unrelated command.

Demo contract

```
store.create({ title: "  Write clear prompts  " })
```

must return:

```
{
  id: 1,
  title: "Write clear prompts",
  status: "pending",
  priority: "normal"
}
```

Weak, useful, reviewable

1. `Implement create.`
2. `Implement TodoStore.create using the test.`
3. A prompt with behavior, file limits, and a check to run.

Compare what the model must still guess in each request.

A change request should change the prompt

Priority adds:

- allowed values;
- a default;
- list filtering;
- an invalid-input decision.

State the new behavior before asking for another change.

A reusable prompt pattern

```
Change <file and scope> to achieve <observable behavior>.
Use <relevant context>.
Keep <constraints and non-goals>.
Verify with <command or review>.
```

State enough detail to make the task testable without dictating the implementation.

Lab handoff

You will extend one todo API:

1. implement the baseline store;
2. wire the REST routes;
3. compare context with one file open and closed;
4. decide invalid-priority behavior;
5. apply the change request;
6. save one reusable prompt.

Deliverable: working code, six passing tests, and a prompt decision log.