Context Engineering Fundamentals

What This Lesson Covers

How to engineer the information Claude receives - organizing context across CLAUDE.md files, skills, and external documents so Claude works with precision on complex projects.

The Core Insight

Context is the bottleneck, not intelligence. Claude is already smart enough - most failures come from missing or poorly organized context. Context engineering is giving Claude exactly the information it needs, when it needs it, without overwhelming the context window.

```
PROMPT ENGINEERING:
"Fix the bug in the login function"


CONTEXT ENGINEERING:
Project rules + code patterns + examples + relevant docs + focused prompt
→ Working solution on first try
```

The Context Hierarchy

Claude loads CLAUDE.md files from multiple locations, merged in this order:

Priority	Location	Scope
1	~/.claude/CLAUDE.md	All sessions, all projects
2	Root CLAUDE.md	Current project
3	Parent directory CLAUDE.md	Monorepo sections
4	Child directory CLAUDE.md	Subdirectory overrides

Key principle: Context cascades. Higher-level rules apply broadly; lower-level rules override for specific contexts.

What Goes in Each CLAUDE.md

Level	What to Include	Example Content
User (~/.claude/)	Personal preferences, universal patterns	Coding style, communication tone, tools you always use
Project (root)	Tech stack, conventions, common commands	"TypeScript + React", "Use kebab-case", "npm test"
Parent directory	Section-specific rules for monorepos	Backend conventions in /server/CLAUDE.md
Child directory	Folder overrides, specialized context	Test patterns in /__tests__/CLAUDE.md

Keep each under 200 lines. For every line, ask: "Would removing this cause Claude to make mistakes?" If not, cut it.

Tools vs Skills vs Workflows

Understanding the distinction:

Concept	What It Is	Example
Tool	Script, MCP server, or API that connects Claude to external services	Context7 MCP, Brave Search, GitHub CLI
Skill	Instructions that tell Claude how to use tools for specific purposes	Research skill combining Context7 + Brave
Workflow	Multi-step process combining multiple skills and tools	Investigate → research → implement → test

Tools give Claude capabilities. Skills tell Claude how to use them. Workflows chain skills together.

Example: You have Context7 (library docs) and Brave Search (web) as MCP tools. A research skill tells Claude to use Context7 first for library-specific questions, fall back to Brave for general patterns, and store findings in docs/RESEARCH.md before implementing.

Progressive Disclosure

Don't load everything upfront. Use index files to let Claude discover what it needs.

The Three Levels

1. **Name** - File/folder name tells Claude what's there
2. **Index** - Entry point file summarizes contents
3. **Content** - Actual examples, patterns, scripts

In Skills

```
skills/research/
├── SKILL.md              ← Entry point (always visible)
├── library-lookup.md     ← Loaded when doing library research
├── web-research.md       ← Loaded when doing web research
└── examples/
    └── research-output.md
```

SKILL.md acts as index - Claude reads it first, then navigates to what it needs.

In Any Folder

Apply the same pattern everywhere:

```
docs/
├── INDEX.md              ← Lists what's in this folder
├── architecture.md
├── api-contracts.md
└── decisions/
    ├── INDEX.md          ← Lists decisions
    ├── auth-approach.md
    └── database-choice.md
```

Claude reads INDEX.md, finds what's relevant, loads only that.

Externalizing Memory

Claude's context resets between sessions. External documents persist.

What to Externalize

Document	Purpose
docs/PLANNING.md	Architecture decisions, feature specs
docs/RESEARCH.md	Investigation findings, Q&A from sessions
docs/BUGS.md	Encountered bugs and their causes
docs/FIXES.md	Solutions that worked
docs/PATTERNS.md	Established patterns to follow
docs/STYLEGUIDE.md	Design decisions, UI conventions

When to Externalize

Tell Claude to save learnings:

- "Add this to RESEARCH.md so we remember"
- "Document this bug and fix in BUGS.md"
- "Update PATTERNS.md with this approach"

This moves knowledge out of ephemeral context into persistent storage Claude can reference later.

Context Breadcrumbs

Higher-level documents should point to lower-level ones, creating a navigation tree.

In CLAUDE.md

```
# Project Rules


## Architecture

See @docs/PLANNING.md for system design


## API Patterns

Follow examples in @examples/api-handlers/

For authentication: @docs/decisions/auth-approach.md


## When Working On...

- Frontend components: @src/components/README.md

- Database changes: @docs/migrations.md

- Testing: @__tests__/TESTING.md
```

The Branching Tree

```
CLAUDE.md (root)

├── Points to → docs/PLANNING.md

├── Points to → examples/INDEX.md

│   └── Points to → specific examples

├── Points to → src/components/README.md

│   └── Points to → component-specific docs

└── Points to → skills (via descriptions)

    └── Point to → nested skill files
```

Claude follows breadcrumbs based on the current task, loading only relevant branches.

Documentation Lookup Skills

Create skills that combine tools for research:

```
---
name: lookup-docs
description: Research library documentation. Use when implementing
features with unfamiliar APIs.
---

## Tools Available
- Context7: Query library documentation
- Brave Search: Find implementation examples
- GitHub CLI: Clone reference repos

## Process
1. Check Context7 for official docs
2. If insufficient, search web for examples
3. If complex, clone a reference implementation
4. Summarize findings in docs/RESEARCH.md before implementing
```

This skill tells Claude *how* to use available tools together effectively.

Querying GitHub Repos

Learn from existing implementations:

```
---
name: research-implementation
description: Research how others solve similar problems.
Use when building complex features.
---


1. Use gh search repos to find relevant projects
2. Clone promising ones to /tmp for analysis
3. Read structure, identify patterns
4. Note approaches that fit our conventions
5. Document findings before implementing
```

Session Persistence

Progress Files

Track work across sessions:

```
# PROGRESS.md

## Current: User Authentication

### Completed
- [x] Database schema
- [x] API routes

### In Progress
- [ ] Frontend forms (validation started)

### Decisions Made
- Using bcrypt (see docs/decisions/auth.md)
- Following patterns from examples/user-crud/

### Next Session
1. Complete form validation
2. Add error handling
3. Write tests
```

Handoff Skills

Create skills for session boundaries:

```
---
name: handoff
description: Prepare context for next session or subagent
---


Before ending work:
1. Update PROGRESS.md with current state
2. Note any discoveries in docs/RESEARCH.md
3. Document blockers or questions
4. List specific next steps
5. Commit with descriptive message
```

Works for session breaks AND for handing off to subagents.

The Examples Pattern

Examples are context engineering's most powerful tool.

Structure

```
examples/
├── INDEX.md              ← What's here and when to use it
├── api-endpoints/
│   ├── simple-crud.ts
│   └── authenticated-route.ts
├── components/
│   ├── form-with-validation.tsx
│   └── data-table.tsx
└── screenshots/          ← UI examples
    ├── form-states.png
    └── error-displays.png
```

Using Examples

Reference in prompts:

```
Build a user settings page following examples/components/form-with-
validation.tsx
```

Screenshots for UI: When building interfaces, include screenshots showing expected visual output. Claude can reference them for layout, spacing, and component styling.

Quick Reference

Pattern	Purpose
CLAUDE.md hierarchy	Layer context by scope
Index files	Enable progressive disclosure
Tools	Connect to external services
Skills	Define how to use tools
Workflows	Chain skills together
@path/file	Import into CLAUDE.md
docs/ folder	Externalize persistent memory
Breadcrumbs	Point to related context
PROGRESS.md	Track work across sessions
examples/	Reference implementations + screenshots

Try It: Set Up Context Engineering

4. Create root CLAUDE.md with breadcrumbs pointing to docs/
5. Create docs/INDEX.md listing your documentation files
6. Create one skill that combines multiple tools
7. Add an examples folder with INDEX.md

Test with a prompt that requires navigating the context tree.

What's Next

You've learned to engineer Claude's context. Next lesson: orchestrating multi-phase projects that span many sessions and involve multiple agents.