

# Prompt Engineering Cheatsheet

---

## Claude Code Training | Quick Reference

---

### The Golden Rule

---

**Be clear, direct, and specific.** Claude performs best when you tell it exactly what you want.

❌ "Can you maybe help me with some code stuff?"

✅ "Refactor the user authentication module to use JWT tokens instead of sessions."

---

### XML Tags Structure

---

Use XML tags to organize complex prompts:

```
<context>
You are reviewing a Node.js REST API that handles user authentication.
The codebase uses Express, PostgreSQL, and follows REST conventions.
</context>

<task>
Add input validation to the POST /api/users endpoint.
Validate: email format, username (3+ chars), password (8+ chars).
</task>

<requirements>
- Use a validation library (suggest one if needed)
- Return 400 Bad Request with specific error messages
- Add corresponding unit tests
</requirements>
```

```
<output_format>
Show the modified route handler and new test file.
</output_format>
```

## Common Tags

Tag	Purpose	Example
<code>&lt;context&gt;</code>	Background information	Tech stack, constraints
<code>&lt;task&gt;</code>	What you want done	The specific request
<code>&lt;requirements&gt;</code>	Constraints and criteria	Must-haves, avoid list
<code>&lt;examples&gt;</code>	Input/output samples	Show the pattern
<code>&lt;output_format&gt;</code>	How to structure response	JSON, markdown, code

## Chain of Thought

Ask Claude to think step-by-step for complex problems:

```
Before implementing the caching layer, think through:
1. What data needs caching?
2. What's the cache invalidation strategy?
3. What are the failure modes?

Then provide your implementation.
```

**Trigger phrases:** - "Think step by step..." - "Walk me through your reasoning..." -  
"Before you answer, consider..." - "Let's break this down..."

## Thinking Modes (Claude Code)

Phrase	Thinking Level	Use When
"think"	Low	Simple analysis, quick questions
"think hard"	Medium	Multi-file changes, design decisions
"think harder"	High	Complex refactoring, architecture
"ultrathink"	Maximum	Critical decisions, security review

### Example:

```
Think harder about how to implement rate limiting
across our microservices without a central store.
```

## Few-Shot Examples

Show Claude the pattern you want:

```
<examples>
<example>
Input: "usr_abc123"
Output: { "type": "user", "id": "abc123" }
</example>

<example>
Input: "org_xyz789"
Output: { "type": "organization", "id": "xyz789" }
</example>
</examples>

<task>
```

```
Now parse: "team_def456"  
</task>
```

---

## Role Assignment

---

Set Claude's perspective:

```
You are a senior security engineer reviewing code  
for vulnerabilities. Focus on:  
- SQL injection  
- XSS attacks  
- Authentication bypasses  
- Exposed secrets
```

```
Review this pull request with that lens.
```

**Effective roles:** - Senior [role] at a [type] company - Expert in [domain] with 10+ years experience - Code reviewer focused on [aspect] - Technical writer creating docs for [audience]

---

## Output Control

---

### Length Control

- "Be concise" / "Brief response"
- "Provide a comprehensive analysis"
- "In 3 bullet points..."
- "In under 100 words..."

### Format Control

- "Respond in JSON format"
- "Use markdown with headers"
- "Create a table comparing..."
- "List as numbered steps"

## Tone Control

- "Explain like I'm a junior developer"
  - "Use technical language appropriate for architects"
  - "Keep it casual and friendly"
- 

## Avoiding Hallucinations

---

### Do

```
Based ONLY on the code in this repository,  
explain how authentication works.
```

### Don't

```
How does authentication work in this codebase?  
(Claude might invent details)
```

## Grounding Techniques

1. **Limit scope:** "Only use information from these files..."
  2. **Ask for citations:** "Quote the relevant code..."
  3. **Request uncertainty:** "If you're not sure, say so..."
  4. **Verify facts:** "After your response, list any assumptions you made."
- 

## Iterative Refinement

---

Start broad, then narrow:

```
Round 1: "Give me an overview of how this API handles errors"
```

```
Round 2: "Focus on the error handling in the payment module"
```

Round 3: "Show me how to add retry logic to the payment error handler, following existing patterns"

---

## Common Patterns

---

### Code Review

```
<task>Review this code for:</task>
<criteria>
- Security vulnerabilities
- Performance issues
- Maintainability concerns
- Test coverage gaps
</criteria>
<format>
For each issue:
1. Location (file:line)
2. Severity (Critical/High/Medium/Low)
3. Description
4. Suggested fix
</format>
```

### Bug Investigation

The /api/users endpoint returns 500 errors intermittently.  
Error in logs: "Connection timeout"

Think through possible causes, then:

1. List diagnostic steps
2. Identify the most likely root cause
3. Propose a fix

## Feature Implementation

```
<feature>
Add password reset functionality via email
</feature>

<constraints>
- Use existing email service (src/services/email.js)
- Tokens expire in 1 hour
- Rate limit: 3 requests per hour per email
- Follow existing route patterns
</constraints>

<deliverables>
1. New route handlers
2. Database migration for reset tokens
3. Unit tests
4. Integration test
</deliverables>
```

## Documentation

```
Generate API documentation for the user endpoints.
Include:
- Endpoint path and method
- Request body schema (if applicable)
- Response schema with examples
- Error responses
- Authentication requirements

Format as OpenAPI 3.0 YAML.
```

---

## Anti-Patterns (What NOT to Do)

⚡ Anti-Pattern	⚡... Better Approach
"Make this code better"	"Refactor for readability, add error handling"
"Is this good?"	"Review for [specific criteria]"
"Fix bugs"	"The login fails when email has '+'. Find and fix."
"Write tests"	"Write unit tests covering edge cases for validateEmail()"
Long, rambling context	Use XML tags, be structured
Assuming Claude knows your stack	State tech stack explicitly

## CLAUDE.md Quick Template

```
# Project Context

## Tech Stack
- [Language] + [Framework]
- [Database]
- [Testing framework]

## Commands
- `npm run build` - Build project
- `npm test` - Run tests
- `npm run lint` - Check code style

## Conventions
- [Coding style rules]
- [Naming conventions]
- [File organization patterns]
```

```
## Do
- [Preferred patterns]
- [Required practices]

## Don't
- [Anti-patterns to avoid]
- [Deprecated approaches]
```

---

## Quick Reference Card

---

### Prompt Structure

1. **Context** - What Claude needs to know
2. **Task** - What you want done
3. **Constraints** - Rules and limitations
4. **Format** - How to structure the response

### Power Phrases

- "Think step by step..."
- "Based only on the code in..."
- "Before implementing, outline your approach..."
- "If uncertain, ask clarifying questions..."

### Debugging Prompts

- "Why might this be failing?"
- "What edge cases am I missing?"
- "Walk me through the execution flow..."

### Quality Prompts

- "Review this for security issues..."
  - "How would a senior engineer improve this?"
  - "What's the simplest solution that works?"
-

