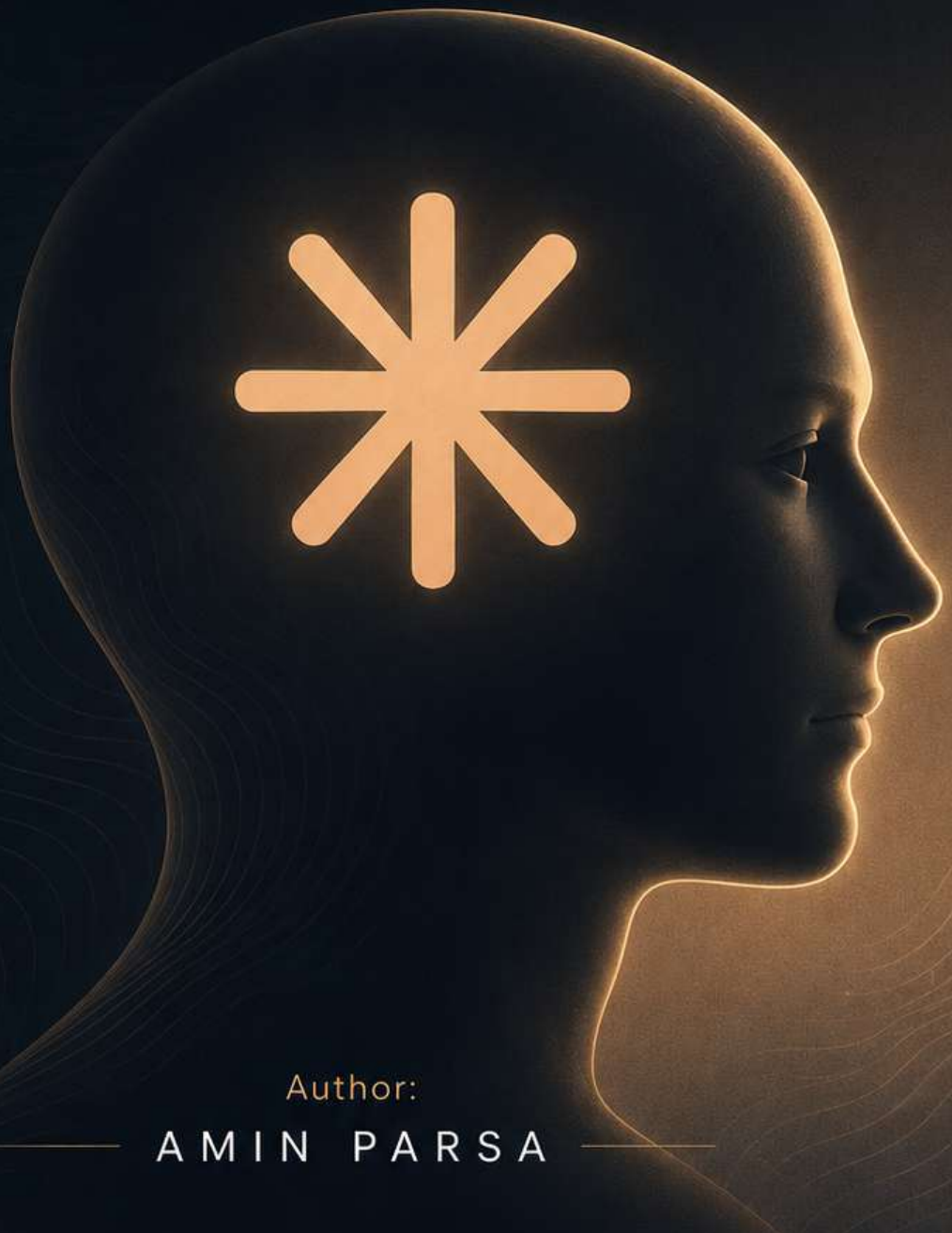
THE COMPLETE GUIDE TO

# Claude

## FROM **ZERO** TO **PROFESSIONAL**

Prompt Engineering, Context Management,
and Advanced Thinking



PROMPT ENGINEERING

CONTEXT MANAGEMENT

COWORK & PROJECTS

SKILLS & AUTOMATION

Author:
AMIN PARSA

# The Complete Guide to Claude: From Zero to Professional

**Prompt Engineering, Context Management, and Advanced Thinking**

*Written in 2026, a strange and early era for a technology that's still becoming itself. This edition is a snapshot — dedicated to how much better things are about to get.*
*Dedicated to the AI that helped write about itself. May this book be hilariously outdated within the year.*

---

## Table of Contents

## Part Four: Cowork and Integrated Management



## Part Five: Claude for Content Creation and Writing



## Part Six: Claude for Research and Academia



## Part Seven: Skills and Automation



## Part Eight: Advanced Techniques

---

# Introduction

## The Problem Most Users Have When Using AI

If you've ever used ChatGPT, Gemini, or Claude, you've probably run into this problem: you write a long prompt, but the output doesn't match what you expected. The reason is simple: you need to work with AI like a professional colleague, not like a search engine.

Many users think it's enough to ask a question and wait for a complete answer. But the reality is that AI tools deliver their best results when *you* specify the context of the work, explain the goal, and define the output format. This is exactly what you'll learn in this book.

## Why Claude Is Built for Serious Work

Claude was built by Anthropic, and compared to ChatGPT — which is optimized for general conversation — its main focus is specialized, in-depth work. Claude can read multiple files at once, analyze long texts, and help you with complex projects.

**Claude's Key Features:**

- **Deeper reasoning:** suited for complex, multi-step analysis
- **Better Context management:** can hold a large amount of information in memory and draw on it
- **A Cowork environment for professional work:** a structured workspace for long-term projects
- **Professional code generation:** optimized for programmers and developers

If you want to work on a real project — not just ask a quick question — Claude is a good choice.

## Who Is This Book For?

This book is for anyone who wants to use Claude professionally. Programmer, writer, researcher, or project manager — it doesn't matter. If you have repetitive tasks, if you want to speed up your work, or if you're looking for a tool to analyze and generate content, this book is for you.

**Primary Audiences**

- **Programmers:** for generating code, fixing bugs, optimizing, and documenting
- **Writers and content creators:** for writing, editing, and generating creative content
- **Academic researchers:** for analyzing papers, summarizing, and writing theses
- **Project managers:** for organizing information, generating reports, and automating tasks
- **Anyone who wants to automate repetitive work**

## How to Use This Book and the Reading Order

This book is designed so you can choose your reading path based on your level and needs. You don't have to start from the beginning.

| If you are... | Start here |
|---|---|
| A beginner | Read the chapters in order |
| A programmer | Go straight to the Programmers section |
| Working with Cowork | Part Four is the key to your success |
| A content creator or writer | See the Prompt Writing and Content chapters |
| An academic researcher | See the Research and Academia section |
| Looking to automate repetitive tasks | See the Skills and Automation section |

Every chapter includes a concept explanation, sample prompts, real-world examples, and common mistakes. You can copy the prompts directly and use them in your own work.

---

# Part One: Claude's Architecture and Model Selection

# Chapter 1: The Claude Ecosystem — More Than Just a Chatbot

## Introduction

Claude isn't just a simple chatbot; it's a complete ecosystem designed for project management, coding, content generation, and automation. Many users only ever use the Chat section, and in doing so miss out on a large part of what Claude can actually do. In this

chapter, you'll get to know the six main sections of the Claude ecosystem and learn which section to use for which task.

## 1. Chat: A Place to Start, Not a Place to Stay

Chat is the simplest and most accessible part of Claude. It's suited for quick, exploratory work:

- Asking a quick question
- Testing a small idea
- Drafting an initial piece of text

**Limitations of Chat**

- **Temporary context:** once you close the page, the entire conversation is lost.
- **No custom instructions:** you can't set Global Instructions.
- **Not suited to serious projects:** it isn't enough for long-term or complex work.

If you want to work on a project that spans several days or weeks, Chat is not the right choice.

## 2. Projects: Persistent Context for Long-Term Work

Projects is Claude's professional workspace, designed for serious projects.

**Key Differences from Chat**

- **Persistent context:** all conversations and files are saved.
- **Custom Instructions:** you can set custom instructions so Claude works exactly in your style.

**Use Cases for Projects**

- Writing a book or a long article
- Research and data analysis
- Generating serialized content
- Recurring workflows (e.g., weekly content production)

If you have a project that will take several days or weeks, Projects is the best choice.

## 3. Code: A Professional Coding Environment

Code is an environment designed for developers that lets you work directly on your local files.

**Key Differences from Chat**

- **Reading and writing local files:** Claude can read and edit your project's files.
- **Direct code editing:** no need to copy-paste; Claude modifies the code directly.

**Use Cases for Code**

- Working on a real codebase
- Debugging and fixing bugs
- Refactoring code
- Writing tests
- Automatic documentation

If you're a developer and want Claude to help on a real project, Code is the best option.

## 4. Cowork: The Most Powerful Part of Claude

Cowork is a combination of Projects and Code, and is considered the most powerful part of the Claude ecosystem.

**Main Difference from Other Sections**

- **Direct access to local files with no need to upload them:** Claude can read, edit, and manage your files and folders.

**Use Cases for Cowork**

- Working on serious, complex projects
- Building custom workflows
- Writing a book or multiple chapters using local files
- Managing project folders

If you want Claude to be a full participant in your project, Cowork is the ideal choice.

## 5. Skills: Custom Add-ons for Repetitive Tasks

Skills let you build custom skills or add-ons for Claude.

**Use Cases for Skills**

- Automatically generating articles in a specific format

- Converting text into a social media post
- Running a repetitive workflow

By using Skills, Claude goes from being a general-purpose tool to a personal assistant that works exactly in your style.

## 6. Connectors: Connecting to External Tools

Connectors let Claude connect to your favorite tools:

- **Google Drive:** reading and analyzing documents
- **Slack:** reviewing and summarizing messages
- **Gmail:** summarizing emails
- **Notion:** consolidating project information

**Use Cases for Connectors**

- Automatically summarizing daily emails
- Analyzing documents stored in Google Drive
- Reviewing Slack messages and extracting key points
- Consolidating project information from Notion

With Connectors, Claude plugs directly into the heart of your workflow.

## Summary: Do Each Task in the Right Place

Now that you're familiar with the six main sections of the Claude ecosystem, here's where each kind of task belongs:

- **Chat:** quick, exploratory conversations
- **Projects:** long-term work with persistent context
- **Code:** coding and software development
- **Cowork:** real projects with access to local files
- **Skills:** custom commands and capabilities
- **Connectors:** connecting to external services

In the next chapter, you'll learn how to choose the right Claude model, and which model is better suited to which task.

---

# Chapter 2: Choosing the Right Model

## Introduction

One of the first decisions you have to make when working with Claude is which model to use. Claude has three main models: Opus, Sonnet, and Haiku. These three models are, in effect, different versions of the same system, but they differ in their level of reasoning, response speed, and suitability for different kinds of work. Choosing the right model can dramatically improve the quality of your output.

## Opus

Claude Opus is the most powerful model in the Claude family. It's suited to work that requires deep thinking, complex analysis, and multi-step reasoning. If you want to write a specialized article, debug complicated code, or design a business strategy, Opus is the right choice. Its main advantage is greater precision and depth, but in exchange it's usually slower than the other models and is overkill for simple tasks.

## Sonnet

Claude Sonnet is the middle, balanced model. It offers a mix of speed and quality and is suited to most everyday work. Writing blog articles, summarizing text, answering general questions, writing simple scripts, and even moderate coding are all good use cases for Sonnet. In many scenarios, Sonnet is the best default choice, because it offers both good quality and efficient speed.

## Haiku

Claude Haiku is the fastest and lightest model in the Claude family. It's designed for simple, fast, high-volume work. If you want to generate short answers, categorize text, extract simple information, or run automation tasks, Haiku is a good option. Using heavier models for this kind of work is usually a waste of resources.

## A Comparative Example

To better understand the difference between the models, you can compare a single problem across all three:

- **Haiku** gives a fast, short answer.
- **Sonnet** gives a more balanced answer with a brief explanation.
- **Opus** gives a more complete, more analytical answer that examines different angles.

In short:

- If the question is simple → **Haiku**
- If you want a moderate, practical explanation → **Sonnet**
- If you need deep analysis → **Opus**

## Use Cases

Each model is better suited to a particular kind of work.

**Opus for:**

- Writing specialized articles
- Analyzing complex data
- Solving multi-step logical problems
- Advanced coding
- Strategy design
- Reviewing legal or financial documents

**Sonnet for:**

- Blog articles
- Summarizing text
- Translation
- Brainstorming
- General-purpose answers
- Moderate coding
- Editing and rewriting

**Haiku for:**

- Short answers
- Text categorization
- Extracting simple information
- Generating short content
- Automation tasks
- Processing a high volume of messages or data

## Real-World Scenarios

Model choice matters in practical scenarios too:

- For **writing an educational article**, Sonnet is usually the best option.
- For **analyzing a long report**, Opus is more suitable.
- For **generating hundreds of short answers**, Haiku is the better choice.
- For **help with programming**: if the task is simple, use Sonnet; if it's a large, multi-file project, Opus is better.

## Summary

The takeaway from this chapter is that you shouldn't use Claude with one fixed model. Each model is designed for a specific level of work:

- **Haiku** for speed and simple tasks
- **Sonnet** for most everyday work
- **Opus** for complex analysis and deep reasoning

A professional user knows which model to choose for which problem. Making that choice correctly both raises the quality of the output and saves time. The next chapter covers writing effective prompts, because even the best model won't give you the result you want without a clear instruction.

---

# Chapter 3: Extended Thinking

## Introduction

When you ask Claude to solve a complex problem, you have two modes:

1. **Normal mode:** Claude thinks quickly and answers.
2. **Extended Thinking mode:** Claude spends more time analyzing before answering.

The difference between these two modes is like the difference between a doctor who diagnoses you immediately and one who first runs various tests, reviews the results, and then gives a final diagnosis.

## What Is Extended Thinking?

Extended Thinking is a special capability in the Opus and Sonnet models that lets Claude work through its thinking process internally before responding. In this mode, Claude:

- Examines different hypotheses
- Compares alternative solutions

- Checks itself for possible mistakes
- Carries out multi-step reasoning

**Result:** a more precise, more logical, more trustworthy answer.

## A Practical Example: Analyzing a Drop in Sales

Suppose you run an online store and your sales dropped 30% last month. You ask Claude to analyze why.

**Normal Response**

In normal mode, Claude might raise a few generic reasons:

- A technical problem on the site
- A cut in advertising budget
- Increased competition
- A seasonal shift

General suggestion: check Google Analytics.

This answer is generic and lacks deep analysis.

**Extended Thinking Response**

In Extended Thinking mode, Claude first examines different hypotheses:

- If it's a technical problem, the bounce rate should have risen
- If it's an advertising problem, traffic should have dropped
- If it's a competitor problem, the conversion rate should have dropped
- If it's a seasonal problem, a similar pattern should show up from last year too

Then, based on the data, it concludes that:

- Traffic has stayed steady
- The conversion rate has dropped
- So the problem is likely a drop in conversion, not a drop in visits

**Result:** a more precise, more logical, more trustworthy answer.

**Likely causes:**

- A new competitor with lower prices
- A problem in the payment process

- A change to the product page that reduced user trust

**Suggested actions:**

1. Check the payment gateway logs
2. Compare prices with competitors
3. Review recent changes to the product pages
4. Run an A/B test on the checkout page

## When Should You Use Extended Thinking?

### Good fits

Extended Thinking is well suited to work that requires deep analysis:

- Complex analysis
- System design
- Scientific research
- Multi-step reasoning
- Strategic decision-making

### Poor fits

For simple, quick tasks, Extended Thinking is a waste of resources:

- Simple tasks
- General questions
- Repetitive tasks
- Situations where speed matters more than precision

## Key Points

1. Extended Thinking takes more time.
2. It doesn't exist for Haiku — it's only active in Opus and Sonnet.
3. It costs more because it consumes more tokens.
4. You can watch the thinking process in the interface.
5. It's very useful for serious, professional work.

## Summary

Extended Thinking makes Claude think more deeply before answering, which makes it better suited to complex analysis, system design, research, and important decision-making. This capability is the difference between a quick answer and a professional analysis.

The next chapter compares Claude with other AIs like ChatGPT, Gemini, and Copilot.

---

# Chapter 4: Claude Compared to Other AIs

When it comes to artificial intelligence, choosing the right tool can make a huge difference in the quality of your work. Claude isn't just one option among dozens of language models — it's a tool built with a different approach. In this chapter we compare Claude with its three main competitors: ChatGPT, Google Gemini, and Microsoft Copilot.

## ChatGPT: The Pioneer, the All-Rounder

ChatGPT is a product of OpenAI and was the first model to bring conversational AI to the general public. It has been released in several versions, the newest of which is GPT-5.5. ChatGPT performs very well at generating creative content, writing marketing copy, brainstorming, and general conversation. High response speed and the ability to understand a wide range of contexts are among its strengths.

But ChatGPT can run into limitations on complex programming projects or deep technical analysis. Its answers can sometimes be shallow or lack the necessary detail. It can also fall short when a task requires multi-step reasoning or structured analysis.

**Best used for:** quick content generation, writing emails, brainstorming, general conversation, and answering simple questions.

## Google Gemini: Search Power and Multimodality

Gemini is a Google product built on top of Google's powerful search infrastructure. It can access up-to-date information and work with images and video alongside text. Gemini performs well at online research, summarizing information, and tasks that need fresh data.

That said, Gemini isn't as strong as Claude at advanced programming and deep code analysis. It can also lack the precision needed on projects that require holding long context and tracking fine details.

**Best used for:** online research, access to current information, working with images and video, and content summarization.

## Microsoft Copilot: Integration with the Microsoft Ecosystem

Copilot is a Microsoft product deeply integrated with Office tools, Visual Studio, and GitHub. It's extremely useful for people working within the Microsoft environment. Copilot can help in Word, Excel, PowerPoint, and even while coding in VS Code.

But Copilot has limitations outside the Microsoft ecosystem. On independent projects or work that requires deep analysis and complex reasoning, it may not be as capable as Claude.

**Best used for:** working with Microsoft tools, writing code in Visual Studio, and managing documents in Office.

## Claude: A Focus on Deep Reasoning and Programming

Claude is a product of Anthropic, designed with a focus on safety, transparency, and deep reasoning. It excels at code analysis, advanced programming, software architecture, and work that requires multi-step thinking. Claude can hold long context and doesn't lose track of details on complex projects.

One of Claude's distinctive features is Projects, which lets you give the model custom knowledge, files, and specific instructions. This makes Claude very well suited to long-term and team projects.

**Best used for:** advanced programming, code analysis, software architecture, complex projects, and multi-step reasoning.

## A Practical Example: Designing an Authentication System

Suppose you want to design an authentication system for a web application. Let's look at how each model responds:

**Question:** "I want to design a secure authentication system for a Node.js application. What architecture do you suggest?"

**ChatGPT's answer:**
ChatGPT will likely give you a general list of technologies: use JWT, use bcrypt to hash passwords, use Passport.js, and store tokens in localStorage. The answer is quick and useful, but it lacks deep security detail.

**Gemini's answer:**
Gemini might point to online articles and documentation and give you links to authentication best practices. The information is current and useful, but it may offer less practical code.

**Copilot's answer:**
Copilot will likely suggest code using Azure Active Directory or the Microsoft Identity Platform. If you're in the Microsoft ecosystem, this is very useful, but it may be limiting for independent projects.

**Claude's answer:**
Claude first asks questions: Do you need multi-factor authentication? Do you want to use OAuth? How many users are you expecting? Then it proposes a complete architecture that includes multiple layers of security, session management, token refresh, rate limiting, and even sample implementation code. Claude also explains security considerations such as preventing CSRF, XSS, and SQL Injection attacks.

Claude doesn't just write the code — it explains the reasoning behind each decision and helps you understand the architecture. This educational, in-depth approach is extremely valuable on real-world projects.

## Summary

No single model is the best at everything. The right tool depends on your needs:

- If you want to generate content quickly or get ideas, **ChatGPT** is a good choice.
- If you need up-to-date information and online search, **Gemini** is a good fit.
- If you work within the Microsoft environment, **Copilot** offers excellent integration.
- But if you want to write complex code, design architecture, or work on long-term projects, **Claude** is the superior choice.

In the next chapter, we'll learn how to set up your work environment with Claude.

---

# Chapter 5: Setting Up Your Work Environment

Now that you're familiar with Claude and its capabilities, it's time to set up your work environment. In this chapter, you'll learn step by step how to create an account, set up a project, and get the most out of Claude.

## Step One: Create an Account

To start working with Claude, you first need to create an account:

1. Go to claude.ai.
2. Click the **Sign Up** button.

3. You can sign up with your email or your Google account.
4. A confirmation email will be sent to you. Click the confirmation link.
5. Once confirmed, you'll land in the Claude dashboard.

Claude has two types of plans:

- **Free Plan:** limits the number of messages, but is enough to get started.
- **Pro Plan:** more messages, access to more powerful models, and advanced features.

If you want to use Claude seriously in your projects, the Pro plan is recommended.

## Step Two: Get to Know the Interface

When you log into Claude, you'll see a simple, clean interface. Your list of conversations sits on the left, and you can chat with Claude in the middle of the screen.

There are two important sections at the top of the page:

- **Chats:** your regular conversations with Claude.
- **Projects:** custom projects to which you can add specific knowledge and instructions.

## Step Three: Create Your First Project

Projects is one of Claude's most powerful features. By creating a project, you can give Claude custom knowledge, files, and specific instructions that it will draw on across every conversation within that project.

**Steps to create a project:**

1. Click the **Projects** tab.
2. Click the **Create Project** button.
3. Choose a name for the project — for example, "Online Store Project."
4. Write your specific instructions in the **Custom Instructions** section.

**Example instructions:**

```
This project is an online store built with Node.js and React.
- We use TypeScript.
- The backend is written with Express.js.
- The database is PostgreSQL.
- Always write clean, maintainable code.
- Use async/await instead of callbacks.
```

These instructions tell Claude to write code based on these standards across every conversation in this project.

## Step Four: Add Knowledge to the Project

One of Claude's great capabilities is that you can add your own files and documents to a project. Claude reads these files and draws on them in its answers.

**Steps to add Knowledge:**

1. On the project page, go to the **Project Knowledge** section.
2. Click **Add Knowledge**.
3. Upload your files. You can upload text files, PDFs, code, and even API documentation.

**Example:** if you have a `database-schema.sql` file that shows your database structure, upload it. Now when you ask Claude to write a query, it knows exactly what your tables and columns are called.

## Step Five: Start a Conversation Within the Project

Now that you've built the project and added knowledge, you can start working:

1. Click on your project.
2. In the chat box, write your question or request.

**Example:**

```
Write an API for user registration that receives an email and password,
hashes the password, and saves it to the database.
```

Claude writes the appropriate code based on the instructions and knowledge you added to the project.

## Step Six: Install the Claude CLI (Optional)

If you want to use Claude directly in your terminal or code editor, you can install the Claude CLI.

**Installation steps:**

1. Go to the **Settings** page in Claude.
2. Go to the **API Keys** section and create an API key.
3. Copy the key and store it somewhere safe.

**Install the CLI:**

```
npm install -g @anthropic-ai/claude-cli
```

**Set the API key:**

```
export ANTHROPIC_API_KEY="your-api-key-here"
```

Now you can use Claude directly in the terminal:

```
claude "Write a function to compute a factorial"
```

## Important Tips for Getting Started

- **Be clear:** the more precise your request, the better Claude's answer.
- **Give context:** if Claude knows about your project, it writes better code.
- **Iterate:** if the first answer isn't complete, ask your question more precisely.
- **Use Projects:** for long-term projects, always make use of the Projects feature.

## Summary

In this chapter, you learned how to create an account, set up a project, add custom knowledge, and get the most out of Claude. Now you're ready to move on to more advanced techniques and build real projects in the chapters ahead.

---

# Part Two: Principles of Professional Prompt Writing

# Chapter 6: The Logic of Prompt Writing

Most users, when working with Claude, treat it like a search engine: they ask a question and expect a precise answer back. But Claude isn't a search engine — it's a reasoning system. That means the quality of its answer depends heavily on how you phrase your request.

For this reason, prompt writing isn't simply "writing an instruction" — it's designing a problem for an intelligent system.

In this chapter, we'll first look at how Claude interprets a prompt, then examine the problems caused by ambiguous instructions, and finally build up the structure of an effective prompt.

## How Claude Interprets a Prompt

When you write a prompt, Claude doesn't see it as just a simple sentence. The model tries to extract several things from your text:

- The goal of the task
- The context of the problem
- The type of output expected

If this information isn't present in the prompt, Claude is forced to guess at it.

For example, consider this prompt:

*"Write an article about artificial intelligence."*

This instruction is very generic. Claude doesn't know:

- What the exact topic is
- Who the audience is
- How long the article should be
- Whether the tone should be formal or casual

As a result, the answer will usually be generic and shallow.

Now let's make the same request more precise:

*"Write an article of about 800 words on the impact of artificial intelligence on education. The audience is high school teachers. The tone should be formal but easy to understand. The structure should include an introduction, three main benefits, and a conclusion."*

In this case, Claude knows exactly what it needs to produce. The result is usually more structured, more precise, and more usable.

The simple rule is: the more information you give Claude, the more likely it is to produce a suitable answer.

## The Problem with Ambiguous Instructions

One of the most common reasons for getting weak answers from AI is the use of ambiguous instructions. An ambiguous instruction is one that can be read in several different ways.

**Example 1: "Optimize my code"**

This sentence could mean several things:

- Improving speed
- Improving code readability
- Reducing memory usage
- Rewriting the program's structure

Claude doesn't know which of these is actually your goal.

**Example 2: "Write an email"**

The problem with this instruction is that it's missing key information:

- Who the email is for
- What the purpose of the email is
- Whether the tone should be formal or friendly
- Whether the email should be short or detailed

As a result, Claude produces a generic email that will likely need a lot of editing.

**Example 3: "Summarize this text"**

Here too there are several ambiguities:

- How many lines the summary should be
- Whether it should extract only the main points, or add explanation too
- Who the summary is being written for

The main lesson: when an instruction is ambiguous, the model is forced to guess. The more it has to guess, the more likely the result is to miss what you actually needed.

## The Structure of an Effective Prompt

A good prompt usually has three main parts:

**1. Goal**

What needs to be done?

Examples:

- "Write a Python function"
- "Send a follow-up email"
- "Analyze this data"

**2. Context**

What situation is the problem set in?

Examples:

- "The function should take a list of numbers and calculate their average"
- "This email is a follow-up on a project we haven't heard back about in 2 weeks"
- "This data is about the monthly sales of an online store"

**3. Expected Output**

What should the result look like?

Examples:

- "The code should be readable, handle errors, and include a usage example"
- "The email should be formal but friendly and no more than 100 words"
- "The analysis should include the sales trend, the best month, and actionable recommendations"

**The simple formula for a professional prompt:**

**Goal + Context + Expected Output = Effective Prompt**

This structure helps Claude understand much more precisely what you want.

## Practical Example: Before and After Refining the Prompt

Let's walk through a real example.

**Weak prompt:**

*"Write code to calculate a factorial."*

Problems:

- The programming language isn't specified
- The input type isn't specified
- Error handling isn't specified
- The output structure isn't specified

As a result, the answer will usually be just a simple function.

**Professional prompt:**

Goal: Write a Python function to calculate a factorial.

Context:

- The function should take a positive integer
- It should return an error if the input is negative or non-numeric
- The function should be optimized for large numbers (up to 1000)

Expected output:

- Clean, readable code
- Includes a docstring
- Includes a usage example
- Includes a unit test

In this case, Claude knows exactly:

- Which language to use
- How to handle the input
- What structure the output should have

As a result, the answer will usually be more complete and more usable.

## Summary

Understanding the logic of prompt writing is the first step toward using Claude professionally. In the next chapter, we'll learn how to design short but highly effective prompts using a simple formula.

---

# Chapter 7: The Short Prompt Formula

In the previous chapter, we learned that an effective prompt needs three parts: a goal, context, and an expected output. Now we're going to turn that concept into a practical formula you can use in any situation.

This formula helps you get professional results even with short prompts.

## The Structure: Goal + Context + Expected Output

The short prompt formula looks like this:

**[Goal] + [Context] + [Expected Output]**

These three parts can be expressed in just a few simple sentences, but their impact is huge.

**Goal**

What needs to be done?

Examples:

- "Write an email"
- "Optimize this code"
- "Prepare a summary"

**Context**

The information Claude needs in order to do the task.

Examples:

- "This email is a follow-up on a project we haven't heard back about in 2 weeks"
- "The code should be optimized for speed, not readability"
- "The summary is for a management report"

**Expected Output**

What should the final result look like?

Examples:

- "The email should be formal but friendly, no more than 100 words"
- "The code should keep its current structure"
- "The summary should be no more than 5 lines and include only the key points"

When you put these three parts together, you get a complete prompt.

**Final example:**

*"Write a follow-up email. This email is for a project we haven't heard back about in 2 weeks. The tone should be formal but friendly, and no more than 100 words."*

This prompt is short, but it has all the information it needs.

## Turning Weak Prompts into Professional Prompts

Let's turn a few weak prompts into professional ones.

**Example 1: A content-writing request**

Weak prompt:

*"Write an Instagram post."*

Problems:

- The topic isn't specified
- The audience isn't specified
- The tone isn't specified
- The length isn't specified

Professional prompt:

*"Write an Instagram post about the benefits of morning exercise. The audience is people aged 25 to 35 who have recently started working out. The tone should be motivational and friendly. The text should be no more than 150 words and include 3 relevant hashtags."*

**Example 2: A coding request**

Weak prompt:

*"Write an API."*

Problems:

- The programming language isn't specified
- The type of API isn't specified
- The functionality isn't specified
- The output structure isn't specified

Professional prompt:

*"Write a REST API in Node.js and Express that fetches a list of users from the database and returns it as JSON. The API should include error handling and return an appropriate error message if the database is unavailable. The code should be readable and include comments."*

**Example 3: An analysis request**

Weak prompt:

*"Analyze this data."*

Problems:

- The type of analysis isn't specified
- The purpose of the analysis isn't specified
- The expected output isn't specified

Professional prompt:

*"This is monthly sales data for an online store. Analyze it and tell me which months had higher sales, what patterns exist, and what suggestions you have for increasing sales. Format the output as a short report with 3 sections: sales trend, patterns, and recommendations."*

**Example 4: A summarization request**

Weak prompt:

*"Summarize this article."*

Problems:

- The length of the summary isn't specified
- The audience isn't specified
- The type of summary isn't specified

Professional prompt:

*"This is a scientific article about climate change. Write a summary of no more than 200 words that's understandable for environmental science students. The summary should include the main findings and the final conclusion."*

**Example 5: A translation request**

Weak prompt:

*"Translate this text."*

Problems:

- The target language isn't specified
- The tone of the translation isn't specified
- The type of text isn't specified

Professional prompt:

*"This text is a formal email in English. Translate it into Persian. Keep the tone formal and professional. Where technical terms exist, use the appropriate Persian equivalent."*

**Example 6: A debugging request**

Weak prompt:

*"My code doesn't work."*

Problems:

- The exact problem isn't specified
- No error message is provided
- The expected behavior isn't specified

Professional prompt:

*"This Python code is supposed to read a list of files from a folder, but it throws a 'FileNotFoundError'. The folder path is correct and the files exist. Check the code and tell me where the problem is and how to fix it."*

**The pattern common to all these examples:**

- A weak prompt is just a generic instruction
- A professional prompt includes a goal, context, and an expected output

## Hands-On Practice

Now it's your turn to practice this formula. Below are a few weak prompts. Try rewriting them using the Goal + Context + Expected Output formula.

**Exercise 1**

*"Design a logo."*

Questions that need to be added to the prompt:

- What business is the logo for?
- What style should it have?
- What colors should be used?
- What format should the output be in?

A better sample prompt:

*"Give me an idea for a logo for an online programming education startup. The logo style should be modern and minimal. The main colors should be blue and white. Explain what concept the logo conveys."*

**Exercise 2**

*"Give me a workout plan."*

Questions that need to be clarified:

- What's the goal of the workout?
- What's the person's fitness level?
- How many days a week will they train?
- Is the workout at home or at the gym?

A better sample prompt:

*"Design a 4-day workout program for building physical strength. The audience is a beginner who trains at the gym. Each session should be no more than 45 minutes, and the exercises should include a brief explanation."*

**Exercise 3**

*"Give me a business idea."*

Problems:

- The field isn't specified
- The target market isn't specified
- The capital level isn't specified

A better sample prompt:

*"Suggest 3 online business ideas that can be started with a small amount of capital. The target market is Iran, and the focus should be on digital services. For each idea, write a short*

*explanation of the revenue model."*

**Exercise 4**

*"Write an article."*

A better sample prompt:

*"Write an article of about 1000 words on the benefits of learning to code for teenagers. The audience is parents. The tone should be educational and simple, and the article should include an introduction, three main benefits, and a conclusion."*

The goal of these exercises is to build a mental habit: before sending a prompt, ask yourself:

1. What exactly is the goal?
2. What information does Claude need?
3. What should the output look like, exactly?

Once you keep these three questions in mind, writing professional prompts becomes a natural skill.

## Summary

In the next chapter, we'll learn how to use examples to teach Claude exactly what type of answer to produce.

---

# Chapter 8: Using Examples to Teach the Model

One of the most effective techniques in prompt writing is to show rather than explain. Claude and other language models learn from patterns. When you provide one or more examples, the model can pick up on your style, structure, and tone, and produce similar output.

This technique is called Example-Driven Prompting.

In this chapter, we'll learn how to use examples to teach Claude.

## The Concept of Example-Driven Prompting

Example-Driven Prompting means that instead of telling the model *how* to do something, you show it *what* you want.

**The key difference:**

**Descriptive method:**

*"Write an Instagram post. The tone should be friendly, use emojis, keep the sentences short, and end with a question."*

**Example-driven method:**

*"Write an Instagram post. Like this:*

*'Had an amazing cup of coffee this morning ☕*
*Sometimes it's these small moments that make the day.*
*What about you? What made you feel good today?'"*

With the second method, Claude understands exactly:

- How informal the tone should be
- Where the emojis should go
- How short the sentences should be
- How the question should be framed

## Why Are Examples Better Than Descriptions?

**1. They're more precise**

Describing a "friendly tone" means something different to everyone. But when you give an example, there's no ambiguity.

**2. They're faster**

Writing an example takes about 30 seconds. Writing a precise description might take 5 minutes.

**3. Better learning**

Claude learns from patterns. The more examples you give, the more precise the output becomes.

## How Many Examples Should You Give?

- For simple tasks: 1 example is enough
- For complex tasks: 2 to 3 examples work better
- For highly specialized tasks: 3 to 5 examples are recommended

General rule: the more variety there is among your examples, the better Claude understands the overall pattern.

## Example: Teaching a Writing Tone

Writing tone is one of the hardest things to describe. But with an example, it becomes very simple.

**Scenario:** You want Claude to write a LinkedIn post for you, but your tone is distinctive.

**Weak prompt:**

*"Write a LinkedIn post about the importance of continuous learning. The tone should be professional but warm."*

Problem: "professional but warm" means something different to everyone.

**Prompt with an example:**

*"Write a LinkedIn post about the importance of continuous learning. The tone should match these previous posts of mine:*

*Example 1:*
*'Three years ago I wrote my first line of Python code.*
*Today I'm managing a team of 10.*
*The secret? Learn something new every day. Even if it's small.'*

*Example 2:*
*'A junior developer asked me a question yesterday that I didn't know the answer to.*
*Instead of feeling embarrassed, I sat down with them and we learned it together.*
*Learning isn't a one-way street.'*

*Now write a similar post about the importance of continuous learning."*

**Result:** Claude now knows:

- Sentences should be short and direct
- Personal experience should be used
- The tone should be warm but not overdone
- Informal punctuation should be used

**Another example: A formal tone**

If you want a completely formal tone:

*"Write an email to the CEO. The tone should match this previous email of mine:*

*'Dear Sir/Madam,*
*Following a review of the Q1 report, I would suggest holding a meeting with the sales and marketing teams to discuss ways to improve performance.*
*Please let us know a suitable time if you agree.*
*Regards,*
*[Name]'*

*Now write a similar email requesting an additional budget."*

With this method, Claude knows exactly what level of formality you want.

## Example: Teaching an Output Structure

Sometimes what matters isn't what's written, but how it's written.

**Scenario:** You want Claude to prepare a daily task list, but you have a specific structure in mind.

**Weak prompt:**

*"Make me a daily task list."*

**Prompt with an example:**

*"Make me a daily task list. The structure should be exactly like this:*

*Morning (8-12):*
*- [ ] Check emails (30 min)*
*- [ ] Team meeting (1 hour)*
*- [ ] Work on Project A (2 hours)*

*Afternoon (13-17):*
*- [ ] Reply to tickets (1 hour)*
*- [ ] Write the weekly report (1 hour)*

*Evening (17-19):*
*- [ ] Read a specialized article (30 min)*

*Now make a similar list for tomorrow."*

**Result:** Claude now knows:

- Tasks should be grouped by time block

- Each task should have an estimated duration
- Checkboxes should be used
- The structure should have three sections: morning, afternoon, evening

**Another example: A report structure**

*"Write a weekly project-progress report. The structure should be like this:*

*This week's progress:*
*- Completed the homepage design*
*- Implemented the registration system*

*Challenges:*
*- Delay in receiving data from the design team*
*- Issue connecting to the API*

*Next week's plan:*
*- Complete the payment module*
*- Start initial testing*

*Now write a similar report for the current project."*

With this method, Claude understands that the report needs three sections, and what kind of information each section should contain.

## Using Examples in Code Generation

Example-Driven Prompting is also extremely powerful in programming.

**Scenario:** You want a new function written that's consistent with your project's coding style.

**Weak prompt:**

*"Write a function to retrieve user info by email."*

**Prompt with an example:**

*"Write a function to retrieve user info by email. The coding style should match this existing function:*

```
function getUser(id) {
  return db.query('SELECT * FROM users WHERE id = ?', [id]);
}
```

*The new function should look up by email."*

**Result:** Claude now knows:

- The programming language is JavaScript
- It should use the `db.query` pattern
- Parameters should be passed as an array
- Naming should follow the same style

**Another example: Commenting style**

*"Write a function to calculate a discount. The commenting style should be like this:*

```
def calculate_tax(amount):
    """
    Calculate the tax based on the amount.

    Args:
        amount (float): the base amount

    Returns:
        float: the tax amount
    """
    return amount * 0.09
```

*The new function should calculate the discount."*

With this method, Claude doesn't just write the code — it documents it in the same style, too.

## Summary

Example-Driven Prompting is one of the most powerful techniques in prompt writing. It rests on a simple principle: showing is better than explaining.

When you provide an example, the model can pick up on the exact patterns of tone, structure, and style. This makes the output more consistent, more predictable, and closer to what you actually expected.

In many cases, a short example can replace several sentences of explanation. That's why many professional users, instead of writing long prompts, prepare a set of good examples ahead of time and pull from them when needed.

**Key takeaways from this chapter:**

- Provide 1 example for simple tasks, 2-3 for complex tasks, and 3-5 for specialized tasks
- Examples should be varied so the model can grasp the overall pattern
- Use examples to teach tone, structure, format, and even coding style
- A good example can replace several paragraphs of explanation

In the next chapter, we'll cover something that matters even more than the prompt itself: Context. Understanding Context properly and how to manage it is one of the key skills for using Claude professionally.

---

# Chapter 9: Managing Context

One of the most important differences between beginner and professional users of AI models is how they manage Context. Many people assume the quality of the output depends only on the instruction (the prompt), when in practice, Context often plays a bigger role than the instruction itself.

Context means all the information the model has available to understand your situation: information about the project, the goal, the audience, your working style, constraints, and even personal preferences. If this information isn't present, the model is forced to guess. The result of that guessing is usually a generic, low-value answer.

In contrast, when Context is provided properly, even a very short instruction can produce a precise output. In this chapter, we'll look at why Context matters and how to manage it professionally.

## Why Context Matters More Than the Instruction

Imagine asking a new colleague at your company to write some text for a client. If you just say "write an email to the client," they have very little to go on. They don't know who the client is, what stage your relationship is at, what the company's tone is, or what the email is for.

AI models run into exactly the same problem.

**Example without Context:**

Prompt: *"Write an email to the client and ask for feedback."*

Likely output:

*Hello,*
*Thank you for using our product.*
*Please share your feedback about your experience with us, if you can.*

This text has no particular problem, but it's completely generic. Almost any other company could send the same email.

**Example with Context:**

Prompt: *"We're a software startup that builds project management tools for small teams. The client in question is an 8-person team that bought a subscription two weeks ago. Our brand's tone is friendly and informal. Write a short email asking how their experience has been so far."*

Likely output:

*Hey,*
*It's been two weeks since you started with TaskFlow, and we'd love to know how it's going.*
*Has your team been able to make use of the features?*
*If you've hit any snags or have any suggestions, we'd be happy to hear them.*
*Thanks for trusting us.*

The difference is clear. In the second case, the model knows:

- What kind of business you run
- How big the client's team is
- What stage your relationship is at
- What tone you want

**General rule:** the more complete the Context, the less need there is for complex prompts. Professional users usually spend more time building good Context than writing long instructions.

## Using Files as Context

One of the best ways to manage Context is to use files. Instead of rewriting your project information or working style into the prompt every single time, you can store that information in separate files and hand it to the model whenever it's needed.

This has several important benefits:

- Your information becomes structured
- You can reuse the same Context across multiple projects

- If you're working as a team, everyone can draw on the same shared set of Context

**Example: A project-info file**

Suppose you're working on a software product. You could create a simple file called `project_overview.md`:

```
# Project Info
Project name: TaskFlow
Product type: project management tool for small teams
Primary audience: startups and teams of 5-20 people
Key features: task management, team calendar, progress reports
Tech stack: Backend with Python and Django, Frontend with React
Brand tone: simple, friendly, and direct
```

If this file is available to Claude, many contextual questions disappear on their own. The model knows what product you have, who your audience is, and what tone it should write in.

**Example: A writing-style file**

```
# My Writing Style
- Sentences are short and direct
- No complicated vocabulary
- Warm but professional tone
- Uses real-world examples
- Doesn't overuse emojis

Sample text:
"Three years ago I wrote my first line of code. Today I manage a team.
The difference? Continuous learning."
```

With this file in place, you no longer need to explain how you write every single time.

**Benefits of using files:**

- Write once, use forever
- Your Context is stable and controllable
- You can update the file and the changes apply everywhere
- Your team can use the same files too

## The ABOUT_ME Folder Structure

A useful technique for organizing Context is to create a dedicated folder for your baseline information. Many professional users create a folder called `ABOUT_ME` and put their stable, recurring information there.

This folder usually contains files that describe your working persona, your projects, and your preferences.

**Suggested structure:**

```
ABOUT_ME/
├── profile.txt
├── writing_style.txt
├── projects.txt
└── preferences.txt
```

**1. profile.txt**

General information about you: your field, your areas of expertise, the kinds of projects you work on, and your professional goals.

**2. writing_style.txt**

Describes what your writing style is like — for example, short sentences, an instructional tone, use of practical examples, and avoiding complicated jargon.

**3. projects.txt**

A brief description of your main projects and the areas you're working on.

**4. preferences.txt**

Your personal preferences for working with the model. For example: answers should be short, should include a practical example, and should avoid unnecessary explanation.

When a set of files like this exists, the model can form a much clearer picture of you. As a result, the answers become noticeably more personal and more practical.

This method is especially useful for people who work with AI models continuously — such as writers, researchers, developers, or content creators.

## Example: A Project With Complete Context

To better understand the importance of Context, let's walk through a real scenario.

Suppose you want Claude to help you write blog articles for a software product. If you only give this instruction:

*"Write an article about project management."*

You'll likely get a generic piece of text about project management that closely resembles thousands of other articles on the internet.

But imagine a situation where you've given the model complete Context instead:

**File `product.txt`:**

```
Product name: TaskFlow
Product type: online project management tool
Target audience: small startup teams
Main advantage: simplicity and fast setup
Main competitors: Trello and Asana
```

**File `audience.txt`:**

```
Our audience is usually startup founders, product managers, and small
teams.
They don't have much time to learn complicated tools and are looking for
quick, simple solutions.
```

**File `writing_style.txt`:**

```
The content style is educational and practical. Texts are short, real
examples are used, and complicated management jargon is avoided.
```

Now the instruction becomes very simple:

*"Based on the Context files, write a blog article about why small teams need a simple project management tool."*

In this case, the model has all the information it needs. It knows what your product is, who the audience is, and what tone the content should have. The result is usually a text that's much closer to what you actually needed.

## Summary

This is why many professional users, instead of over-focusing on writing complex prompts, build Context systems instead. When Context is designed properly, even very short instructions can produce precise, professional output.

In the next chapter, we'll move on to more advanced prompt-writing techniques and learn how to use Claude's more advanced capabilities to solve more complex problems.

---

# Chapter 10: Designing Interaction Through Questions

One of the important differences between using an AI model casually and using it professionally is how you design the interaction. Many users assume they need to lay out their entire request in the very first message, and that the model should immediately produce a final answer. But in practice, the best results come when the model asks questions first.

The same thing happens in the real world. If you go to a doctor, a consultant, or a developer, they'll ask you a few questions before offering a solution. These questions help them understand the problem more precisely. AI models perform at their best when they're allowed to go through the same process.

For this reason, one of the important techniques in professional prompt writing is designing interactions in which the model first gathers information and then responds. This pattern applies across many kinds of projects — from content creation to product design, data analysis, and programming.

## Why AI Should Ask Questions

Language models are usually designed to produce an answer even with incomplete information. That is, if they don't have enough data, they'll try to generate an answer by guessing. At first glance this behavior seems useful, but in many cases it produces very generic answers.

For example, imagine giving the model this request:

*"Write a marketing strategy for my product."*

There's very little information in this sentence. The model doesn't know:

- What your product is
- Who your audience is
- What market you operate in
- What your budget is
- What the main goal of the campaign is

Even so, the model will likely produce a generic strategy — something like a combination of social media marketing, content creation, and online ads. Such an answer might be theoretically correct, but it's usually not very useful in practice.

By contrast, if the model asks questions first, the quality of the output improves considerably.

For example, if the model's response is:

*"To design a better strategy, I have a few questions:*
*1. What exactly is your product?*
*2. Who is your primary audience?*
*3. What stage of growth is your business currently at?*
*4. Roughly what is your marketing budget?*
*5. What's the main goal of the campaign? (Increasing sales, acquiring users, or building brand awareness)"*

Now the model is gathering Context. Once you answer these questions, the model can offer solutions that are precisely matched to your situation.

Asking questions has several important benefits:

**1. It increases the accuracy of the answer**

With more information, the model can offer a more precise solution.

**2. It prevents generic answers**

You no longer have to receive a generic answer, reject it, and explain yourself again.

**3. A more natural interaction**

Working with the model starts to feel more like working with a real consultant than using an automated tool.

**4. It helps define the problem precisely**

Sometimes you yourself don't know exactly what you need. The model's questions can help you identify your actual needs.

In many professional projects, the question-asking stage is even more important than producing the answer. When the right questions are asked, the path to the right answer becomes much shorter.

## Designing Prompts That Ask Questions First

For the model to ask questions first, you need to clearly define that behavior in the prompt. If no such request is present in the prompt, the model will usually go straight into producing an answer.

**Technique 1: An explicit instruction to ask questions**

One of the simplest methods is to directly ask the model to pose questions before giving an answer.

*"I want to write an article for my blog. Before you start, ask me a few questions so you can nail down the right topic and angle for the article."*

In this case, the model knows it needs to gather more information first. The result of this interaction is usually a short conversation in which the model raises a few key questions, and after receiving the answers, produces the final text.

**Technique 2: Defining a step-by-step process**

The second method is to break the workflow into stages — that is, tell the model exactly what steps to go through.

*"I want to design a landing page for my product. Follow these steps:*
*1. Ask questions to better understand the product and the audience*
*2. Summarize and analyze the answers*
*3. Write the landing page copy"*

This kind of prompt gives the model a clear structure and results in output that's more organized and precise.

**Technique 3: Using the phrase "before starting"**

The third method is to use phrases like "before starting" or "ask questions first." These phrases give the model a clear signal that it shouldn't jump straight into the answer stage.

*"I want to design an online course. Before offering a proposal, ask a few questions first so we can pin down the course's audience and goal."*

**Technique 4: Defining a consultant role**

A more professional approach is to give the model the role of a consultant. When the model is placed in the role of a consultant, its natural behavior is to assess the situation first.

*"Act as a business consultant. Before offering any suggestion, ask questions so you can better understand my business's situation."*

This kind of interaction design makes conversations with the model feel more natural and more useful.

## Practical Example: Designing an Email Marketing Campaign

To better understand the difference between the two approaches, let's walk through a real example.

**Weak approach (no questions):**

Prompt: *"Write a marketing email for my product."*

Likely output:

*Subject: Introducing our new product*

*Hello,*
*Thank you for being a subscriber to our newsletter.*
*Our new product has unique features that can help you. Click the link below for more information.*

This text is completely generic and has no specific connection to your product or your audience.

**Professional approach (with questions):**

Prompt: *"I want to launch an email marketing campaign. Before writing the email, ask a few questions so you can design an effective one."*

Model's response:

*Sure. To design an effective email, I have a few questions:*
*1. What's your product or service?*
*2. Who's the audience for this email? (Current customers, potential customers, or trial users)*

*3. What's the main goal of this email? (A sale, introducing a new product, or increasing engagement)*
*4. Do you have a specific offer or discount?*
*5. What's your brand's tone? (Formal, friendly, or technical)*

Now suppose you give these answers:

1. We have an online project management tool for small teams.
2. Our audience is product managers and startup founders who signed up already but haven't purchased a subscription yet.
3. The goal is to encourage them to buy a monthly subscription.
4. Yes, we have a 20% discount for the first month.
5. Our tone is friendly and direct.

Final output:

*Subject: Your team is ready. So is TaskFlow.*

*Hey,*
*You signed up a few weeks ago and created a few test tasks.*
*Now it's time to take it a step further.*
*With TaskFlow you can:*
*Split tasks among your team members*
*See project progress in real time*
*Use automatic reminders*
*To make it easier to get started, we've set up a 20% discount for the first month.*
*Just click the link below and activate your subscription.*
*[Get started with 20% off]*
*If you have any questions, just reply here. We're always online.*

The difference is clear. The second email is written precisely for your audience, has a clear goal, and its tone matches your brand.

## Summary

In practice, many professional users, instead of writing one long prompt, create a multi-stage conversation with the model. In the first stage, information is gathered; in the second stage, analysis is done; and in the third stage, the final output is produced.

When this pattern is used properly, the model stops being just an answer-generating tool — it becomes a thinking partner that can understand the problem, ask the right questions, and

ultimately deliver a more precise solution.

In the next chapter, we'll move on to more advanced prompt-writing techniques and learn how to use more complex structures to solve multi-step problems.

---

# Chapter 11: Common Mistakes in Prompt Writing

In the previous chapters, we covered the principles and techniques of professional prompt writing — from prompt structure to using examples, managing Context, and designing interaction. But in practice, many users still get weak output even when they know these principles. The main reason is usually a handful of simple but common mistakes.

Language models are very powerful, but they're just as dependent on the quality of the instruction. A vague or incomplete prompt can push even the best model into producing generic, low-value answers. Learning the right techniques is only half the journey; the other half is recognizing common mistakes and avoiding them.

In this chapter, we'll first go through 10 common user errors, and then offer a simple checklist you can run through before sending any prompt.

## 10 Common User Mistakes

**1. Overly generic requests**

One of the most common mistakes is that users make very generic requests and expect the model to produce exactly what's in their head.

Bad example:

*"Write an article."*

Problem: the model doesn't know what the topic is, who the audience is, what tone to use, or how long the text should be. The result is usually a generic text that isn't fit for actual use.

Solution: always specify the goal, the context, and the expected output.

*"Write an 800-word article about the benefits of remote work for startup managers. The tone should be professional but easy to understand."*

**2. Forgetting Context**

Context is one of the most important factors in the quality of an answer. Without Context, the model is forced to guess and produce generic answers.

Bad example:

*"Write a sales email."*

Problem: the model doesn't know what the product is, who the audience is, what problem it solves, or what the email's goal is.

Solution: give sufficient background information before your request, or ask the model to ask questions.

*"Write a sales email introducing a project management piece of software. The audience is managers of small teams looking for a simple tool to coordinate their team."*

**3. Expecting mind-reading**

Sometimes users assume the model should know exactly what they want without ever explicitly stating it.

Bad example:

*"Make this text better."*

Problem: "better" could mean what, exactly? Shorter? More formal? Simpler? More engaging? More persuasive?

Solution: specify exactly what change you want.

*"Make this text shorter and give it a friendlier tone."*

**4. Using negative instructions**

Many users tell the model what not to do, but don't say what it should do instead.

Bad example:

*"Write a text that isn't too long and isn't too formal."*

Problem: the model doesn't know exactly what to do. This kind of instruction isn't precise.

Solution: give positive instructions instead of negative ones.

*"Write a 120-word text with a friendly tone."*

**5. Sending very long, unstructured prompts**

Some users think the longer the prompt, the better the result. But long, unstructured prompts usually just confuse the model.

Bad example:

A 300-word paragraph where everything is jumbled together and the model can't tell which parts are the important ones.

Solution: use a clear structure. Label the different sections.

Goal: write a marketing email
Context: our product is a project management tool
Audience: managers of small teams
Expected output: a 150-word email with a friendly tone

**6. Needlessly repeating information**

Sometimes users repeat a piece of information several times in a prompt, thinking it will add emphasis.

Bad example:

*"I want the text to be short. It's really important that it's short. Please keep it short."*

Problem: needless repetition just clutters the prompt without adding any real effect.

Solution: say it once, clearly.

*"Write the text in no more than 100 words."*

**7. Forgetting to use examples**

One of the most powerful ways to teach the model is to provide an example. But many users forget to use this technique.

Bad example:

*"Write a catchy headline."*

Better solution:

*"Write a catchy headline. Something like: 'How to Build a New Habit in 30 Days'"*

By providing an example, the model understands exactly what style and structure you have in mind.

**8. Trying to solve everything in one message**

Some users try to solve the entire problem in one very long prompt. But many problems are better solved in several stages.

Solution: break complex problems into several stages:

- Stage one: gathering information (the model asks questions)
- Stage two: analysis
- Stage three: producing the final output

This approach usually produces better results than a single long prompt.

**9. Ignoring the model's initial response**

Sometimes the model's first response is to ask a question or explain that it needs more information. But the user ignores this response and just repeats the same request again.

Solution: if the model asks a question, answer it. This interaction significantly improves the quality of the final output. If the model is asking a question, it usually means it doesn't have enough Context.

**10. Not using Projects and Knowledge**

Many Claude users re-explain their project information every single time, when they could instead create a Project and store their stable information in the Knowledge section.

Bad example:

Explaining every time that "I have a fintech startup that…"

Solution: write this information once in Project Knowledge. From then on, the model always has this information available, and you no longer need to repeat it.

You can also use files to provide Context. Instead of pasting 500 lines of code into the prompt, upload the file and say: "Review this file and find any potential issues."

## The Professional Prompt Checklist

Before sending any prompt, take a few seconds and run through this checklist:

**1. Is the goal clear?**
☐ Have I specified exactly what output I want?

**2. Have I given enough Context?**
☐ Does the model have the information it needs to understand the problem?

**3. Is the audience specified?**
☐ Have I specified who the output is being written for?

**4. Is the type of output defined?**
☐ For example: an article, a list, code, an email, or an analysis.

**5. Are the constraints specified?**
☐ For example: text length, output format, or language.

**6. Have I given an example, if needed?**
☐ Have I provided a sample of the expected output?

**7. Does the prompt have structure?**
☐ Are the different sections (goal, Context, output) clearly marked?

**8. Does the model need to ask questions before answering?**
☐ Should I ask the model to gather information first?

**9. Have I used files or project Context?**
☐ Have I stored my stable information in Knowledge?

**10. Is the prompt short, clear, and direct?**
☐ Have I avoided repetition and ambiguity?

## Summary

If these points are followed, the quality of the output improves considerably in most cases. In fact, the difference between an average user and a professional one is often not the model they use, but the quality of the prompts they write.

In the next chapter, we'll move on to more advanced prompt-writing techniques and learn how to use more complex structures to solve multi-step problems.

---

# Part Three: Claude for Programmers

# Chapter 12: Using Claude in System Design

One of the most powerful applications of Claude for developers is helping with software architecture design. Before you write even a single line of code, architectural decisions determine how scalable, maintainable, and stable the system will be.

Many developers jump straight into coding without having a complete picture of the system's structure. This approach might not cause problems in small projects, but in real-

world projects it quickly leads to tangled dependencies, code that's hard to maintain, and scalability issues.

Claude can act like a software architect at the system-design stage. You can describe the project's requirements and ask the model to suggest an appropriate architecture, technologies, service structure, and even the database schema.

## A Prompt for Software Architecture Design

For Claude to be able to suggest a good architecture, you need to provide enough information about the system. The more precise this information is, the more useful the output will be.

A good prompt for architecture design usually includes these parts:

**1. System description**

Explain what the system is supposed to do and what problem it solves.

**2. Main requirements**

The capabilities the system needs to have.

**3. System scale**

Number of users, data volume, and traffic level.

**4. Constraints**

Time, budget, or technology constraints.

**5. Non-functional requirements**

Security, performance, reliability, and scalability.

**Sample of a standard prompt:**

```
I want to design the architecture for a software system.

System description:
An online platform for managing team projects.

Main requirements:
- Project and task creation
- Team member management
- File uploads
- Notification system
- Progress reporting

Scale:
- 10,000 active users
- About 50,000 requests per day

Constraints:
- Small development team
- Initial launch time of 4 months

Please:
- Suggest an overall system architecture
- Explain the system's different layers
- Recommend appropriate technologies
- Suggest a database structure
- Explain important security and scalability considerations
```

This kind of prompt makes the model analyze the problem like a software architect, rather than just a code generator.

## Example: Designing a SaaS Product

Suppose you want to build a SaaS project management tool similar to Trello. Claude's response might look something like this:

**Suggested architecture**

Given the small team and limited time, the best approach is to start with a Modular Monolith. This architecture lets the system stay simple, while allowing it to be converted into

Microservices later if it grows.

**Frontend layer**

React + TypeScript

- State management with Redux Toolkit or Zustand
- Real-time communication with WebSocket or Socket.io
- Tailwind or Material UI for the interface

**Backend layer**

- Node.js with Express or NestJS
- PostgreSQL for the primary data
- Redis for cache and sessions
- WebSocket for real-time communication

**Infrastructure**

- Docker for containerization
- AWS S3 for file storage
- AWS EC2, or a simpler platform like Render, for deployment
- Cloudflare for CDN and security

**Backend module structure**

The system can be broken down into a few main modules:

- `auth` : sign-up, sign-in, and authentication
- `users` : user information management
- `projects` : project management
- `tasks` : task management
- `files` : file uploads and management
- `notifications` : sending alerts
- `reports` : reporting and analyzing project progress

Each module usually includes these parts:

- `Controller` : receiving API requests
- `Service` : the core application logic

- `Repository`: database communication
- `Model`: data structure definitions

**Sample project structure:**

```
src/
├── controllers/
├── services/
├── repositories/
├── models/
├── modules/
│   ├── auth/
│   ├── users/
│   ├── projects/
│   └── tasks/
├── config/
└── utils/
```

This kind of structure helps the team keep the code organized and maintainable.

## Analyzing the Generated Architecture

When Claude proposes an architecture, it's better not to accept it purely as a final answer. The real value is in analyzing and improving it.

There are a few important questions to ask when evaluating an architecture:

**1. Is the architecture overly complex?**

Sometimes models suggest very advanced architectures that aren't suitable for a small team.

**2. Are the technologies a good match for the team's skills?**

The "best" technology isn't always the most suitable one for your team.

**3. Is the system scalable?**

If the number of users grows, it should be possible to independently scale parts of the system.

**4. Are the dependencies between modules properly designed?**

Too many dependencies between parts makes future development harder.

One of the best practices is to ask Claude to critique the architecture after receiving it. For example:

*"Review the architecture above and explain its potential weaknesses."*

Or:

*"If the number of users reaches one million, what changes would this architecture need?"*

This kind of interaction turns Claude into an architecture consultant, rather than just an answer-generating tool.

### Summary

In practice, many software teams use Claude at the early stages of system design:

- Brainstorming an initial architecture
- Comparing several different architectures
- Weighing the pros and cons of each option
- Choosing an appropriate structure to start development

The outcome of this process is that before you write any code, you have a clear picture of the entire system's structure, and the likelihood of running into architectural problems down the road is much lower.

In the next chapter, we'll cover how to use Claude to generate actual code and optimize it.

---

## Chapter 13: Generating Code with Claude

One of the most common uses of Claude for developers is generating code. But the difference between a beginner and a professional developer lies in how they use the model to generate that code.

Many developers just give a short instruction and expect to get complete, flawless code back. But in practice, code generated without Context and without precise explanation is usually either incomplete, inconsistent with the project's standards, or has security issues.

In this chapter, you'll learn how to write coding prompts so the output is actually usable in a real project.

## Principles of Professional Code Generation

When you use Claude to generate code, a few simple principles can considerably improve the quality of the output.

**1. Provide the Context of the existing code**

If you're adding code to an existing project, it's best to show the model a sample of the current code. This makes the generated code consistent with your project's writing style, libraries, and structure.

Weak prompt:

*"Write a function for user registration."*

Professional prompt:

*"I have an API built with Express. Here's my current login code:*

*[existing code]*

*Now I want to write an endpoint for user registration that:*
*- Takes an email and password*
*- Hashes the password*
*- Saves the user to PostgreSQL*
*- Returns a JWT token*
*- Uses the same structure and libraries as the code above"*

In this case, the model can produce code that's consistent with your project.

**2. Specify the non-functional requirements**

Code that just "works" isn't enough. In a real project, security, error handling, and performance need to be taken into account too.

Example:

*"Write a function for file upload that:*
*- Only accepts .jpg and .png files*
*- Has a maximum size of 5 MB*
*- Sanitizes the filename*
*- Prevents path traversal*
*- Stores the file in AWS S3*

*- Returns the file's URL*
*- Handles errors"*

With this kind of explanation, the model produces code that's much closer to what a real project actually needs.

**3. Ask the model to explain the code**

Sometimes generated code involves a complex algorithm or logic. In these cases, it's best to ask the model to explain the code.

Example:

*"Write a Python function that calculates the similarity between two texts using cosine similarity, and then explain the logic of the algorithm."*

This helps you actually understand the code you're using, rather than just copying it.

## Practical Example: Building a Product Listing API

Suppose you're building an online store and want to create an endpoint for fetching a list of products.

**Weak prompt:**

*"Write an API for a product list."*

**Professional prompt:**

*"I have an API built with Node.js and Express. The database is PostgreSQL, and I use Prisma. I want to build the following endpoint:*

`GET /api/products`

*Features:*
*- Filter by category*
*- Price filter (minPrice and maxPrice)*
*- Search by product name (search)*
*- Pagination (page and limit)*
*- Sorting (sortBy: price or createdAt)*

*The output should be structured as follows:*

```
{
  "products": [...],
  "pagination": {
    "page": 1,
    "limit": 20,
    "total": 150,
    "totalPages": 8
  }
}
```

*Please:*
*- Write the controller and service code*
*- Add validation*
*- Handle errors"*

**Sample structure of the generated code**

Controller:

```
const productService = require('../services/productService');

exports.getProducts = async (req, res) => {
  try {
    const {
      category,
      minPrice,
      maxPrice,
      search,
      page = 1,
      limit = 20,
      sortBy = "createdAt"
    } = req.query;

    const filters = {
      category,
      minPrice: minPrice ? parseFloat(minPrice) : undefined,
      maxPrice: maxPrice ? parseFloat(maxPrice) : undefined,
      search
    };

    const result = await productService.getProducts(
      filters,
      parseInt(page),
      parseInt(limit),
      sortBy
    );

    res.json(result);
  } catch (error) {
    console.error(error);
    res.status(500).json({ error: "Internal server error" });
  }
};
```

Service:

```
const prisma = require("../prisma");

exports.getProducts = async (filters, page, limit, sortBy) => {
  const where = {};

  if (filters.category) {
    where.category = filters.category;
  }

  if (filters.minPrice || filters.maxPrice) {
    where.price = {};
    if (filters.minPrice) where.price.gte = filters.minPrice;
    if (filters.maxPrice) where.price.lte = filters.maxPrice;
  }

  if (filters.search) {
    where.name = {
      contains: filters.search,
      mode: "insensitive"
    };
  }

  const skip = (page - 1) * limit;

  const [products, total] = await Promise.all([
    prisma.product.findMany({
      where,
      skip,
      take: limit,
      orderBy: { [sortBy]: "desc" }
    }),
    prisma.product.count({ where })
  ]);

  return {
    products,
    pagination: {
```

```
      page,
      limit,
      total,
      totalPages: Math.ceil(total / limit)
    }
  };
};
```

In this example, you can see that when the prompt is written precisely, the model can produce a structure close to a real-world implementation — including separating the controller and service layers, supporting filters, pagination, and error handling.

## Reviewing and Improving the Generated Code

The code Claude generates is usually a good starting point, but it's best to always review it.

A few useful questions to ask when reviewing code:

- Have security issues been taken into account?
- Are inputs properly validated?
- Is the error handling appropriate?
- Is the code readable and maintainable?
- Is its performance suitable for the scale you need?

You can also ask Claude to improve the generated code. For example:

*"Review this code and suggest improvements for security and performance."*

Or:

*"Write unit tests for this endpoint using Jest."*

## Summary

In practice, many developers use Claude as a coding partner: an initial version of the code is generated first, then its quality is reviewed, tests are written, and finally the code structure is refactored.

Used properly, Claude can considerably speed up software development without sacrificing code quality for speed.

In the next chapter, we'll cover how to use Claude for debugging and fixing code issues.

# Chapter 14: Debugging and Fixing Errors with Claude

One of the most time-consuming parts of programming is finding and fixing bugs. Sometimes a small error can take hours to track down, especially when the system's behavior is unpredictable or the error only occurs under specific conditions.

Claude can act as a technical colleague in the debugging process. But as we saw in previous chapters, the quality of the output depends on the quality of your prompt. If you just say "this code doesn't work," the model won't be able to offer a precise analysis either.

In this chapter, you'll learn how to use Claude to identify the root cause of a bug, analyze the error, and get suggested solutions.

## Principles of Professional Debugging with Claude

When you run into a bug, it's best to gather enough information before asking for help.

**1. Define the problem precisely**

A good debugging prompt should include four key elements:

- The relevant code
- The expected behavior
- The actual behavior
- The error message or logs (if any)

**Weak prompt:**

*"This code doesn't work. Why?*
*[code]"*

**Professional prompt:**

*"This function is used to calculate the total of orders:*

*[code]*

*Expected behavior:*
*If there are two orders priced 100 and 200, the output should be 300.*

*Actual behavior:*
*Sometimes it throws a runtime error, and sometimes it returns the wrong result.*

*Error message:*
`TypeError: Cannot read property 'price' of undefined`

*Can you find the problem?"*

With this information, the model can identify the actual cause much faster.

**2. Provide a minimal reproducible example**

One of the best real-world debugging practices is to build a Minimal Reproducible Example — that is, the smallest version of the code that still triggers the same bug. Instead of sending the whole project, send just the relevant part.

Example:

*"The main project's code is large, but the bug occurs in this part:*

```
function processUsers(users) {
  return users.map(user => {
    return {
      id: user.id,
      name: user.profile.name
    };
  });
}
```

*Error:*
`TypeError: Cannot read property 'name' of undefined`

*Sample data:*

```
[
  { id: 1, profile: { name: "Ali" } },
  { id: 2 }
]
```

*How should I write this code more safely?"*

In this case, Claude can quickly spot that some users are missing the `profile` field, and suggest using optional chaining or validation.

**3. Ask the model to explain its analysis process**

Sometimes fixing the bug matters, but understanding why it happened matters even more. You can ask Claude to explain its reasoning.

Example:

*"This code throws an async error:*

*[code]*

*Please:*
*- Explain the cause of the error*
*- Show how the code's execution flow works*
*- Suggest a corrected version"*

This kind of response helps you spot similar bugs faster on your own in the future.

## Practical Example: Fixing a Common Loop Bug

Suppose you have code that calculates the sum of prices:

```
function calculateTotal(items) {
  let total = 0;
  for (let i = 0; i <= items.length; i++) {
    total += items[i].price;
  }
  return total;
}
```

With the following input:

```
[
  { price: 50 },
  { price: 70 }
]
```

This error occurs:

```
Cannot read property 'price' of undefined
```

**Claude's analysis:**

The problem is in the loop condition. The condition `i <= items.length` causes the loop to run one time more than the array's length.

If the array has two elements:

- `items.length = 2`
- The values of `i` will be: 0, 1, 2

But there's no third element, so `items[2]` is `undefined`, and accessing `.price` on it causes the error.

**Corrected version:**

```
function calculateTotal(items) {
  let total = 0;
  for (let i = 0; i < items.length; i++) {
    total += items[i].price;
  }
  return total;
}
```

Or a more modern version:

```
function calculateTotal(items) {
  return items.reduce((sum, item) => sum + item.price, 0);
}
```

This kind of response doesn't just fix the bug — it explains why it happened and suggests a better approach.

## Debugging Performance Issues

Not every bug is a program error. Sometimes the problem is performance.

**Example prompt:**

*"This endpoint takes about 3 seconds to respond:*

*[endpoint code]*

*On every request:*
*- Products are read from the database*
*- Another query is run for each product*

*How can I improve the performance?"*

Claude usually recognizes this kind of issue as the N+1 Query Problem and offers suggestions like:

- Using a `JOIN`
- Eager Loading in the ORM
- Running aggregated queries instead of several separate ones
- Using a cache for frequently accessed data

The goal is to reduce the number of database round-trips and make the processing more efficient.

## Debugging Runtime Errors

Some errors only show up at runtime, and only under specific conditions — for example, under heavy traffic, or when certain data is fed into the system. In these cases, providing logs and the circumstances of the error is very important.

**Example prompt:**

*"This API sometimes throws a 500 error:*

*[endpoint code]*

*Error log:*

```
Error: Connection timeout
  at Database.query (db.js:45)
  at UserService.getUser (userService.js:12)
```

*This error usually happens when traffic is high. How can I resolve this?"*

In a situation like this, Claude might examine several possibilities, such as:

- A limited number of database connections
- A missing or inadequate connection pool
- Queries that are taking too long

- No retry mechanism in place for failures

**Example use of a connection pool:**

```
const pool = new Pool({
  max: 20,
  idleTimeoutMillis: 30000,
  connectionTimeoutMillis: 2000
});
```

**A simple example of retry logic:**

```
async function queryWithRetry(query, retries = 3) {
  for (let i = 0; i < retries; i++) {
    try {
      return await db.query(query);
    } catch (error) {
      if (i === retries - 1) throw error;
    }
  }
}
```

This kind of analysis helps identify the root cause of the problem, instead of just patching the error temporarily.

## Using Claude to Analyze Logs

In real-world systems, you usually deal with a large volume of logs. Manually reviewing these logs is time-consuming, but Claude can help spot patterns in the errors.

Example:

*"Find the cause of the 500 error in these logs:*
*[a section of the server log]"*

The model can:

- Analyze the program's execution path
- Identify recurring error patterns
- Pinpoint where the error originated

- Offer several hypotheses for the root cause

## Claude as a Debugging Partner

Many professional developers don't just use Claude to generate code — they use it as an analysis tool as well. A common workflow might look like this:

1. Explain the bug precisely
2. Send the relevant parts of the code
3. Get an initial analysis
4. Test the suggested solution
5. Ask for further improvement or refactoring

In practice, Claude acts like a second developer who can quickly read the code, form hypotheses, and suggest solutions.

That said, you should always review the results. The best approach is to treat Claude as a tool for analysis, learning, and speeding up the debugging process — not as a full replacement for the software engineering process.

---

# Chapter 15: Code Review and Refactoring with Claude

A large part of programming work isn't writing new code — it's reading, reviewing, and improving code that's already been written. In software teams, this process is known as Code Review. The goal of Code Review is to find bugs, spot security issues, improve design quality, and make future maintenance easier.

Claude can play the role of a fast, precise technical colleague in this process. By handing Claude a piece of code, you can get several different kinds of analysis within seconds: a security review, a performance review, a readability review, an architecture review, and refactoring suggestions.

This tool isn't a replacement for human code review, but it can make the initial review stage much faster and surface many issues before a pull request is ever sent.

## Reviewing Code with Claude

For better results, when you send code, specify what kind of analysis you want. The most common types of review cover security, performance, and code structure quality.

**Security review**

One of the most important uses of Claude in Code Review is finding security vulnerabilities. The model can spot issues like SQL Injection, XSS, CSRF, insecure password storage, or the use of hardcoded secrets.

**Sample prompt:**

*"Review this endpoint for security and explain any potential issues:*

*[code]"*

**Example:**

```
app.post('/api/users', async (req, res) => {
  const { email, password, role } = req.body;
  const user = await db.query(
    `INSERT INTO users (email, password, role) VALUES ('${email}',
'${password}', '${role}')`
  );
  res.json({ success: true, userId: user.id });
});
```

This code has several serious problems:

- Possible SQL Injection due to the use of string interpolation
- Password is stored in plain text
- No input validation
- No error handling

**A safer version:**

```
const bcrypt = require('bcrypt');

const hashedPassword = await bcrypt.hash(password, 10);

await db.query(
  'INSERT INTO users (email, password, role) VALUES ($1, $2, $3)',
  [email, hashedPassword, role]
);
```

Email validation, a minimum password length, and error handling should also be added to the endpoint.

**Performance review**

Sometimes code is logically correct but isn't performance-optimized. Claude can identify common performance-issue patterns.

**Sample prompt:**

*"Review this code for performance and suggest improvements."*

One common backend issue is the N+1 Query problem:

```
const users = await db.getUsers();

for (const user of users) {
  const orders = await db.getOrdersByUser(user.id);
  user.orders = orders;
}
```

In this case, a separate query is run for every single user. If there are 100 users, 101 queries will run. The usual fix is to use a `JOIN` or fetch the data in a single query.

## Refactoring Code

Refactoring is the process of improving a code's internal structure without changing its external behavior. Its goal is to increase readability, reduce complexity, and make the code easier to maintain over time. Claude can be very useful in identifying problematic code and suggesting a better structure.

**Identifying problematic code**

The first step in refactoring is spotting the code's weak points. Before asking for a rewrite, it's best to ask Claude to analyze the code first.

**Code analysis prompt:**

*"Review this code and identify its problems in terms of the following:*
*- Excessive complexity*
*- Code duplication*
*- Poor naming*
*- Violations of SOLID principles*
*- Readability issues*

*Don't rewrite the code. Just explain and prioritize the issues."*

**Sample original code:**

```
def process_user_data(data):
    if data['type'] == 'premium':
        if data['status'] == 'active':
            if data['payment'] == 'completed':
                discount = 0.2
                price = data['price'] * (1 - discount)
                if data['country'] == 'US':
                    tax = price * 0.08
                elif data['country'] == 'UK':
                    tax = price * 0.20
                elif data['country'] == 'DE':
                    tax = price * 0.19
                else:
                    tax = price * 0.15
                total = price + tax
                return {'total': total, 'discount': discount}
    elif data['type'] == 'basic':
        if data['status'] == 'active':
            price = data['price']
            if data['country'] == 'US':
                tax = price * 0.08
            elif data['country'] == 'UK':
                tax = price * 0.20
            elif data['country'] == 'DE':
                tax = price * 0.19
            else:
                tax = price * 0.15
            total = price + tax
            return {'total': total, 'discount': 0}
    return None
```

**Problems with this code:**

- Excessive nesting of conditionals, which reduces readability
- Duplicated tax-calculation logic in several places
- A single function with multiple different responsibilities
- Use of unnamed constant numbers (magic numbers)

- Generic naming, like `data`

**Refactoring example**

Now you can ask Claude to implement the same behavior with a better structure.

**Prompt:**

*"Refactor the code above following these principles:*
*- Reduce the nesting of `if` statements*
*- Separate responsibilities into independent functions*
*- Remove code duplication*
*- Use named constants instead of magic numbers*
*- Improve naming*

*The code's behavior must not change."*

**Sample refactored code:**

```
TAX_RATES = {
    "US": 0.08,
    "UK": 0.20,
    "DE": 0.19
}
DEFAULT_TAX = 0.15
PREMIUM_DISCOUNT = 0.20


def get_tax_rate(country):
    return TAX_RATES.get(country, DEFAULT_TAX)


def calculate_tax(price, country):
    return price * get_tax_rate(country)


def calculate_discount(user_type):
    if user_type == "premium":
        return PREMIUM_DISCOUNT
    return 0


def is_user_active(user):
    return user.get("status") == "active"


def is_premium_payment_valid(user):
    return user.get("payment") == "completed"


def process_user_data(user):
    if not is_user_active(user):
        return None

    user_type = user.get("type")
    price = user.get("price")
```

```
    country = user.get("country")

    if user_type == "premium" and not is_premium_payment_valid(user):
        return None

    discount = calculate_discount(user_type)
    discounted_price = price * (1 - discount)
    tax = calculate_tax(discounted_price, country)
    total = discounted_price + tax

    return {
        "total": total,
        "discount": discount
    }
```

Several important improvements have been made in the new version:

- The main function's complexity has been reduced
- The tax and discount logic has been moved into separate functions
- Code duplication has been removed
- Magic numbers have been replaced with meaningfully named constants
- Each function has a single, clear responsibility

## Practical Use of Claude in Refactoring

In real-world projects, you can use Claude for tasks like:

- Breaking large functions into several smaller ones
- Turning duplicated code into a shared utility
- Improving variable and function naming
- Suggesting appropriate design patterns
- Preparing code for test writing

**Sample prompt:**

*"This file has gotten too large. Split it into several logical modules and explain what responsibility each part has."*

## An Important Note

Even though Claude can offer good suggestions, refactoring should always be carefully reviewed. In real projects, it's best to:

- Have unit tests in place before refactoring
- Make changes incrementally
- Verify the system's behavior after the change

Claude is a powerful tool for improving code structure, but the final decisions about architecture and system design should still be made by the developer.

---

# Chapter 16: Testing and Documentation

In many software projects, code quality doesn't just depend on how it's written — it also depends on how testable and how understandable it is. The two main tools for achieving this are testing and documentation. That said, writing complete tests and precise documentation is usually time-consuming, and many teams keep putting it off.

Claude can make this process much faster. By providing code, a developer can ask Claude to generate suitable tests, or create technical documentation usable by the team or by users.

In this chapter, we'll learn how to use Claude to generate professional-grade tests and documentation.

## Generating Unit Tests

Unit tests are used to verify the behavior of a function or module under different conditions. A good test should cover several types of scenarios:

- Normal cases
- Edge cases
- Invalid inputs
- Errors and exceptions

By analyzing a function, Claude can suggest all of these scenarios.

**Sample prompt:**

*"Write complete unit tests for this function. Cover the following scenarios:*
*- Normal cases*
*- Boundary values*

*- Invalid inputs*
*- Possible errors*

*Use pytest, and make the test names clear."*

**Sample code:**

```
def transfer_balance(balance, amount):
    if amount <= 0:
        raise ValueError("amount must be positive")
    if amount > balance:
        raise ValueError("insufficient balance")
    return balance - amount
```

**Generated test:**

```
import pytest


def test_transfer_normal():
    assert transfer_balance(100, 30) == 70


def test_transfer_full_balance():
    assert transfer_balance(100, 100) == 0


def test_transfer_small_amount():
    assert transfer_balance(50, 1) == 49


def test_transfer_zero_amount():
    with pytest.raises(ValueError):
        transfer_balance(100, 0)


def test_transfer_negative_amount():
    with pytest.raises(ValueError):
        transfer_balance(100, -10)


def test_transfer_insufficient_balance():
    with pytest.raises(ValueError):
        transfer_balance(50, 100)
```

In this example, Claude has covered all the important cases: the normal scenario, the boundaries, and the errors.

## Building Tests for Larger Code

In real-world projects, you're usually working with large classes or modules. In these situations, it's better to ask Claude to first design the test scenarios, and only then generate the test code.

**Better sample prompt:**

*"First list the important test scenarios for this class. Then write a pytest test for each scenario."*

This approach makes the output more structured and easier to review.

## Generating API Tests

Claude can also write tests for APIs. This is especially useful for projects using FastAPI, Flask, or Express.

**Sample prompt:**

*"Write a test for this endpoint that:*
*- Sends an HTTP request*
*- Checks the response status code*
*- Validates the returned data*

*Use pytest and httpx."*

**Sample endpoint:**

```
@app.get("/users/{user_id}")
def get_user(user_id: int):
    user = database.get(user_id)
    if not user:
        raise HTTPException(404, "user not found")
    return user
```

**Generated test:**

```
import pytest
from httpx import AsyncClient


@pytest.mark.asyncio
async def test_get_existing_user():
    async with AsyncClient(base_url="http://test") as client:
        response = await client.get("/users/1")
        assert response.status_code == 200
        data = response.json()
        assert "id" in data
        assert "name" in data


@pytest.mark.asyncio
async def test_get_nonexistent_user():
    async with AsyncClient(base_url="http://test") as client:
        response = await client.get("/users/9999")
        assert response.status_code == 404
```

## Generating Code Documentation

Documentation is one of the areas where Claude is extremely useful. Many projects lack proper explanations for their functions, classes, or modules.

You can ask Claude to:

- Write docstrings for functions
- Generate a project README
- Produce an explanation of the system architecture
- Write an API usage guide

**Sample prompt:**

*"Write a complete docstring for this code, including:*
*- An explanation of the function's behavior*
*- The parameters*
*- The return value*
*- A usage example"*

**Sample code:**

```
def calculate_shipping(weight, distance):
    price_per_km = 0.5
    weight_factor = 1 + (weight / 10)
    return distance * price_per_km * weight_factor
```

**Generated documentation:**

```
def calculate_shipping(weight, distance):
    """
    Calculate the shipping cost based on the package weight and
distance.

    Parameters:
        weight (float): The package weight in kilograms.
        distance (float): The shipping distance in kilometers.

    Returns:
        float: The total shipping cost.

    Example:
        >>> calculate_shipping(5, 100)
        75.0
    """
    price_per_km = 0.5
    weight_factor = 1 + (weight / 10)
    return distance * price_per_km * weight_factor
```

## Generating a README for a Project

One of Claude's very useful applications is generating a README file. Many GitHub repos are hard to use simply because they lack proper explanations.

**Sample prompt:**

*"Write a README for this project, including:*
*- A project description*

*- Installation instructions*
*- How to run it*
*- A usage example*
*- The folder structure"*

If you also provide the project's structure, Claude can generate a much more precise README.

### An Important Note on Using AI for Testing

Even though Claude can generate good tests, they shouldn't be used without review. A few important points:

- Make sure the tests actually verify the expected behavior
- AI sometimes doesn't know project-specific scenarios
- Generated tests can be treated as a starting point

Claude is more like a developer's assistant that can handle a large share of the repetitive work, but the final decision still rests with the developer.

In the next chapter, we'll cover one of the important challenges of software development: working with large, multi-thousand-line codebases, and how Claude can help in understanding and analyzing such projects.

---

## Chapter 17: Working with Large Codebases

In real-world projects, developers rarely work with a single small file or a few hundred lines of code. More often, they're facing systems with thousands, or even millions, of lines of code, written by different people, and changed over the course of years. In situations like this, the biggest challenge isn't just writing new code — it's understanding code that's already been written.

Claude can act as a code analyst here. Instead of spending hours reading through different files, you can hand Claude the important parts of the project and have it explain the system's structure, dependencies, and execution flow to you.

In this chapter, we'll learn how to use Claude to understand, analyze, and work effectively with large codebases.

### Quickly Understanding a Project's Structure

When you join a new project, the first step is understanding its overall architecture. Knowing which part is responsible for the API, which part handles the business logic, and which files are the application's entry points matters a great deal.

You can hand Claude the project's folder structure and ask it to analyze it.

**Sample prompt:**

*"Analyze this project structure and explain:*
*- What each folder's responsibility is*
*- What the project's key files are*
*- What the overall system architecture is*
*- Which file is the application's entry point*

*Project structure:*

```
project/
  src/
    api/
    controllers/
    services/
    models/
    middlewares/
    utils/
    config/
  tests/
  docs/
  package.json
  server.js
```"*

Claude's response will usually be something like this:

*This project is likely a Node.js application with a layered architecture.*

- *The `controllers` folder is responsible for receiving requests and sending back responses.*
- *The `services` folder holds the core application logic.*
- *The `models` folder is used for database communication.*
- *`middlewares` contains shared code such as authentication or logging.*
- *The `server.js` file is likely the application's entry point, which starts the HTTP server.*

An analysis like this helps you grasp the system's overall picture very quickly.

### Analyzing the Application's Execution Flow

In large projects, understanding how a request moves through the system is very important. For example, when a user sends a request to create an order, what paths does that request go through, and which parts of the

```
system get involved?

Claude can explain this path step by step.

**Sample prompt:**

*"When a user sends a `POST /orders` request, explain the code's
execution flow:*
*- Which file the request first goes to*
*- What middlewares run*
*- Which controller gets called*
*- What services get used*
*- What database operations happen*

*[relevant code]"*

The response will usually be something like this:

**Request execution flow:**

1. The request enters the `routes/orders.js` file
2. The authentication middleware runs
3. The `createOrder` function in `orderController` is called
4. The controller validates the data
5. Then `OrderService.createOrder` runs
6. The service saves the information into the `Order` model
7. The result goes back to the controller and a JSON response is sent

This kind of analysis is very useful for understanding complex systems.

### Finding the Code Related to a Feature

Sometimes you need to know where a specific feature is implemented in
the project. For example, you might want to know which files contain the
discount system, the payment system, or the authentication system.

You can ask Claude to identify the files related to a feature.
```

**Sample prompt:**

*"Find the code related to the discount system in this project. Point out the main files and functions.*

*[the project structure or a few important files]"*

The response will usually include something like this:

**Files related to the discount system:**

- `services/discountService.js`: the `calculateDiscount` function
- `controllers/checkoutController.js`: applies the discount at checkout
- `models/couponModel.js`: the model for the coupon codes

This helps you quickly figure out which parts you need to look at to change a feature.

### Analyzing Dependencies

In large codebases, a small change can affect many parts of the system. Before changing an important function, it's best to know where it's used. Claude can help analyze these dependencies.

**Sample prompt:**

*"What parts of the project use this function? If I change it, which parts might be affected?*

*[the function's code]"*

Claude can identify the files that use this function, and even suggest which tests should be run.

### Summarizing Long Files

One common problem in large projects is very long files. Sometimes a single file has several thousand lines of code, and reading the whole thing takes a lot of time.

In situations like this, you can ask Claude to produce a summary of the file.

**Sample prompt:**

*"Summarize this file and explain:*
*- What the file's main purpose is*
*- What its important classes and functions are*
*- Which parts matter for future changes*

*[the file's code]"*

Within seconds, you can get a picture of what that file does.

### Using Claude to Learn New Projects

When you join a new team, it usually takes a few weeks to get familiar with the project's structure. Using Claude intelligently can considerably cut down that time.

Instead of just reading the code, you can:

- Analyze the project's structure
- Review the execution flow of requests
- Identify the important dependencies
- Summarize complex files

In effect, Claude can act as a technical guide that helps you get up to speed on a project much faster.

### An Important Note

There's one thing to keep in mind when using Claude to analyze a

```
codebase: the model only analyzes based on the information you give it. If you only provide a few limited files, the analysis will be limited too.

For this reason, it's best to:

- Provide the project's structure
- Send the key files
- Explain the project's context

The more complete the Context, the more precise Claude's analysis will be.

With the end of this chapter, the "Claude for Programmers" part is also complete. In this part, we saw that Claude isn't just a text-generation tool — it can help developers with system design, code generation, debugging, refactoring, writing tests, and even analyzing large projects.

---

## Part Four: Cowork and Integrated Management

## Chapter 18: Introducing Cowork

Up to this point, we've worked with Claude conversationally: we ask a question, get an answer, and the conversation continues. This approach works fine for short-term tasks, but when you want to work on a real project, you need something more.

Cowork is one of Claude's advanced capabilities that lets you set up a shared, persistent working environment with Claude. In this chapter, we'll learn what Cowork is, how it differs from a regular chat, and how to use it to manage real projects.

### What Is Cowork?
```

```
Unlike regular chats, where each conversation is independent and you
have to re-explain the context once it's over, Cowork defines a
persistent project that includes:

- The project's files (code, documentation, data)
- Global Instructions
- A history of interactions and changes
- Connections to external tools (GitHub, Google Drive, and so on)

In plain terms, Cowork is a persistent workspace where Claude has access
to all your project's files, rules, and context, and can work on the
project in a coordinated way.

### The Difference Between Cowork and a Regular Chat

To understand this better, let's compare the two approaches:

| Feature | Regular Chat | Cowork |
|---|---|---|
| Project memory | Limited to a single conversation | Persistent and
shared across all conversations |
| File access | Must be uploaded every time | Files are stored in the
project |
| Instructions | Must be repeated in every prompt | Global Instructions
are set once |
| Connecting to tools | Not available | Can connect to GitHub, Slack,
and so on |
| Change management | Not available | A history of file changes is
recorded |

**Practical example:**

- **Regular chat:** every time you want Claude to review some code, you
have to upload the file again and explain your coding rules again.
- **Cowork:** you add the files once, write your rules in Global
Instructions once, and from then on Claude always knows them.
```

```
### Applications of Cowork

Cowork is useful for many different kinds of projects. Here are a few of
the most common uses:

**a) Software development**

- Managing a complete project (Frontend + Backend + Database)
- Refactoring and improving code
- Writing tests and documentation
- Reviewing pull requests

**b) Research and data analysis**

- Analyzing large datasets
- Writing data-processing scripts
- Producing analytical reports

**c) Writing and content creation**

- Writing a book or long-form articles
- Managing multilingual content
- Editing and reviewing text

**d) Project management**

- Creating a roadmap and task list
- Tracking progress
- Coordinating between team members

### The Structure of a Cowork Project

A Cowork project usually has a specific structure. Here's a standard
example:
```

```
MyProject/
├── /src main code
├── /docs documentation
├── tests/ tests
├── data/ data
├── .cowork/
│   ├── instructions.md global instructions
│   └── connectors.json tool-connection settings
└── README.md
```

````
Note: the exact structure depends on the type of project, but having an organized structure is key to success with Cowork.

### A Simple Example: Creating a Cowork Project

Let's learn to work with Cowork through a practical example. Suppose you want to build a simple API for managing tasks (a to-do list).

**Step 1: Create the project**

First, ask Claude to create the project structure:

**Prompt:**

*"Create a Cowork project for building a To-Do API with Node.js and Express. The project structure should include:*
*- `/src` for the code*
*- `/tests` for the tests*
*- `/docs` for the documentation"*

**Step 2: Define Global Instructions**

Now specify the project's rules and standards. These instructions are written once, and Claude will follow them in all future conversations:

```markdown
# To-Do API Project Instructions

## Coding standards:
- Use TypeScript
- All functions must have JSDoc
- Use Prettier for code formatting

## Architecture:
- Controller-Service-Repository pattern
- Use Prisma for the database
````
````

```
## Testing:
- Minimum 80% coverage
- Use Jest
```

**Step 3: Start working**

Now you can ask Claude to start the work:

**Prompt:**

*"Write an endpoint for creating a new task. It should have validation and be saved to the database."*

Based on the project structure and the Global Instructions, Claude generates standardized code and places it in the appropriate files.

## The Benefits of Using Cowork

Using Cowork has a number of benefits:

- **Consistency:** all code and output is produced according to a single, unified standard.
- **Time savings:** there's no need to repeat instructions in every prompt.
- **Better management:** a history of changes and file versioning is recorded.
- **Team collaboration:** team members can work on a shared project together.
- **Tool integration:** direct connections to GitHub, Slack, Notion, and so on.

## When to Use Cowork

Use Cowork if:

- You're working on a long-term project
- You need to follow a specific set of standards
- You want to manage multiple files
- You're working with a team and need coordination

Use a regular chat if:

- You just have one quick question
- You don't need to save context
- Your task is a one-off

## Summary

Cowork is a professional working environment for using Claude that lets you manage complex projects in an organized way. By using project files, global instructions, and connections to external tools, you can multiply your efficiency.

In the chapters ahead, we'll learn more details about project structure, Global Instructions, file management, and connecting tools.

Next chapter: the standard project structure and best practices for organizing files.

---

# Chapter 19: Standard Project Structure

In the previous chapter, we got to know Cowork and saw how to set up a persistent working environment with Claude. Now it's time to learn how to organize a project properly.

Project structure might seem like a small detail at first glance, but it has a direct impact on the quality of Claude's work. An organized structure helps Claude find files faster, understand the context better, and produce more precise output.

## Why Project Structure Matters

When your project has a clear structure, you gain several important benefits:

- **More speed:** Claude knows where the code is, where the tests are, and where the documentation is.
- **Easier collaboration:** team members can understand the project faster.
- **Easier maintenance:** when you need to change something, you know exactly where to go.
- **Scalability:** the project can grow without becoming a mess.

Simple rule: the clearer the project structure, the better Claude can work within it.

## The Basic Structure of a Cowork Project

A standard Cowork project usually has this structure:

```
project-name/
├── .cowork/
│   ├── instructions.md
│   ├── context.md
│   └── connectors.json
├── src/
├── tests/
├── docs/
├── data/
├── scripts/
├── config/
├── README.md
└── package.json / requirements.txt
```

Let's go through each part:

**`.cowork/`**

The folder dedicated to Claude's configuration. This is where the project's global instructions, working context, and tool-connection settings live.

**`src/`**

The project's main code. All application files live in this folder.

**`tests/`**

Unit, integration, and end-to-end tests. Keeping tests separate from the main code helps keep things tidy.

**`docs/`**

Technical documentation, system architecture, and developer guides. Anything that needs explaining.

**`data/`**

Data files such as JSON, CSV, datasets, or processing outputs.

**`scripts/`**

Helper scripts such as database migrations, seeding, or automation tools.

**`config/`**

Settings for different environments (development, staging, production).

**`README.md`**

An overview of the project, how to install and run it, and its overall structure.

## Project Structure for a Backend

For a Node.js-based API, the structure is usually like this:

```
todo-api/
├── .cowork/
│   └── instructions.md
├── src/
│   ├── controllers/
│   ├── services/
│   ├── models/
│   ├── routes/
│   ├── middleware/
│   ├── utils/
│   └── app.ts
├── tests/
│   ├── unit/
│   └── integration/
├── prisma/
├── config/
└── package.json
```

**Separation of responsibilities:**

- `/controllers`: handling requests and responses
- `/services`: the core application logic (business logic)
- `/models`: defining data structures
- `/routes`: defining endpoints
- `/middleware`: intermediate request processing (like authentication)
- `/utils`: helper functions and general-purpose tools

This pattern means that when Claude needs to create a new endpoint, it knows exactly where the controller code goes, where the service goes, and where the route goes.

## Project Structure for a Frontend

For a React project, the standard structure is this:

```
todo-app/
├── .cowork/
├── src/
│   ├── components/
│   ├── pages/
│   ├── hooks/
│   ├── services/
│   ├── store/
│   ├── types/
│   ├── utils/
│   └── App.tsx
├── public/
├── tests/
└── package.json
```

**Explanation of the parts:**

- `/components`: reusable UI elements
- `/pages`: the app's main pages
- `/hooks`: custom React hooks
- `/services`: communication with the API
- `/store`: state management (Redux, Zustand, or similar)
- `/types`: TypeScript type definitions

This structure helps Claude spot the right place for a new component and follow existing patterns.

## Project Structure for Data Science

Data-analysis projects have a different structure:

```
data-project/
├── .cowork/
├── notebooks/
├── src/
│   ├── data/
│   ├── features/
│   ├── models/
│   └── visualization/
├── data/
│   ├── raw/
│   ├── processed/
│   └── results/
├── tests/
└── requirements.txt
```

**Important notes:**

- `/notebooks` : Jupyter Notebooks for initial analysis and experimentation
- `/src` : code usable in the production environment
- `/data/raw` : raw, untouched data
- `/data/processed` : cleaned and prepared data
- `/data/results` : final outputs and reports

This separation lets Claude know which data can be changed and which must remain untouched.

## Structure for Writing Projects

For projects like a book, article, or documentation:

```
book-project/
├── .cowork/
├── chapters/
│   ├── 01-introduction.md
│   ├── 02-basics.md
│   └── 03-advanced.md
├── research/
├── images/
├── templates/
└── README.md
```

**Explanation:**

- `/chapters`: the main chapters of the book or article
- `/research`: sources, notes, and research
- `/images`: images and diagrams
- `/templates`: standard writing templates

This structure helps Claude, when editing or generating text, better understand each chapter's context and its relationship to the rest of the book.

## Principles for Designing Project Structure

A few key principles for designing a good structure:

**1. Clarity**

Folder names should be clear and understandable. Avoid vague names like `/stuff` or `/misc`.

**2. Separation of responsibilities**

Each folder should have one clear job. Don't mix code with tests.

**3. Scalability**

The structure should allow the project to grow. Plan ahead for the future from the very start.

**4. Consistency**

Use a consistent pattern across the whole project. If you have `/controllers` in one place, don't call it `/ctrl` somewhere else.

## When Should You Change the Structure?

Sometimes the project's initial structure is no longer enough as it grows. These signs indicate it's time for a restructure:

- Folders have gotten too cluttered (more than 10-15 files in one folder)
- Finding files has become time-consuming
- Team members are confused
- Claude keeps suggesting the wrong files

In these cases, it's a good time to redesign the structure.

## Summary

A good project structure is one of the most important factors for success with Cowork. When files, folders, and responsibilities are properly organized, Claude can understand the project faster and produce more precise output.

Key takeaway: design your project structure correctly from the very start. Changing the structure midway through a project is much harder than getting it right at the beginning.

In the next chapter, we'll get to know one of the most important components of a Cowork project: Global Instructions — where the rules and standards for the whole project are defined.

---

# Chapter 20: Global Instructions

In the previous chapter, we looked at the standard project structure. Now it's time to get to know one of the most important components of a Cowork project: Global Instructions.

This section acts like the project's constitution and determines how Claude should work.

## What Are Global Instructions?

Global Instructions are a set of rules, standards, and guidelines defined for the whole project. Claude takes these instructions into account when working on any file or task.

In simple terms, Global Instructions specify:

- What technologies the project uses
- What coding standards need to be followed

- What the project's architecture looks like
- What technical constraints and rules exist

These instructions are usually kept in a file called `.cowork/instructions.md`.

## Why Are Global Instructions Necessary?

Without clear instructions, Claude might:

- Use a different coding style in every file
- Not properly follow the project's structure
- Suggest the wrong tools
- Put business logic in the wrong place

But by defining Global Instructions:

- **Output becomes consistent** — all code follows the same style
- **Faster understanding of context** — Claude knows the project's framework without repeated explanation
- **Less need for manual fixes** — generated code is standardized from the start
- **Simpler team collaboration** — every team member draws on the same shared reference

Key takeaway: in professional projects, defining Global Instructions is nearly essential.

## The Main Content of Global Instructions

A good Global Instructions file usually includes these sections:

### 1. Tech Stack

In this section, you specify what languages and tools the project uses.

Example:

```
Primary language: TypeScript
Backend framework: Express.js
ORM: Prisma
Database: PostgreSQL
Cache system: Redis
```

This information means that when generating code, Claude uses the right tools and doesn't offer irrelevant suggestions.

**2. Code Style**

In this section, rules about coding style are defined.

Example:

- Variable names: camelCase
- Class names: PascalCase
- File names: kebab-case
- Use ESLint and Prettier for code formatting
- Use async/await instead of promise chains

These rules help keep the entire project's code in one consistent style, making it easier for the team to read.

**3. Architecture**

In this section, the software's architectural structure is specified.

Example:

- Use a Layered Architecture
- Controllers are only responsible for handling requests and responses
- Business logic goes in Services
- No complex logic should be written in Routes

Suggested structure:

```
src/
├── controllers/
├── services/
├── models/
├── routes/
├── middleware/
└── utils/
```

With this, when adding a new feature, Claude knows exactly where the controller code, the service code, and the route code should go.

**4. Error Handling**

A professional project should have a standard error-handling system.

Sample guideline:

- All errors should be structured
- Use a central middleware for error handling
- Error messages should not expose sensitive system information

Standard error format:

```
{
  "success": false,
  "error": {
    "code": "RESOURCE_NOT_FOUND",
    "message": "The requested resource was not found",
    "statusCode": 404
  }
}
```

This consistency makes the whole API behave predictably.

**5. Validation**

One common project problem is a lack of proper validation.

Suggested guideline:

- All inputs must be validated
- Use Zod for validation
- All schemas should live in the `/schemas` folder
- No data should enter the system without validation

These rules mean Claude won't forget validation when writing a new endpoint.

**6. Testing**

In this section, the project's testing standards are specified.

Example:

- Every service must have a unit test
- Every endpoint must have an integration test

- Use Jest for testing
- Minimum test coverage of 80%

With this, Claude also suggests relevant tests when generating a new feature.

**7. Security**

You can also define the project's security rules in Global Instructions.

Sample items:

- Passwords must never be stored in plain text
- Use bcrypt for hashing
- Endpoints must have authentication
- Use rate limiting to prevent attacks

These guidelines help Claude pay attention to security considerations when generating code.

## A Practical Example of Global Instructions

Here's an example of a real file:

```
# Global Instructions

## Tech Stack
- Node.js
- TypeScript
- Express
- Prisma
- PostgreSQL

## Code Style
- variables: camelCase
- classes: PascalCase
- files: kebab-case
- use async/await

## Architecture
- use layered architecture
- controllers handle request/response only
- business logic goes in services

## Validation
- use Zod
- all request inputs must be validated

## Testing
- use Jest
- write tests for all services

## Security
- passwords must be hashed with bcrypt
- use JWT for authentication
- protect all private routes
```

This file helps Claude follow a clear framework while working on the project.

## Important Notes on Writing Global Instructions

For effective Global Instructions, follow these points:

**1. Write clearly**

Don't use vague sentences. Instructions should be precise and actionable.

Bad: *"The code should be clean."*

Better: *"Use async/await, not promise chains."*

**2. Be short but complete**

Mention the important rules, but avoid writing unnecessary detail.

**3. Keep it updatable**

As the project grows, the instructions may need to change. Review the file regularly.

**4. Keep it realistic and practical**

Write rules that are actually followed in the project. Theoretical instructions that aren't respected just create confusion.

## Summary

Global Instructions is one of the most important tools for managing Cowork projects. These instructions establish the project's overall framework and make sure Claude respects the project's standards when generating code or content.

Simple rule: the more precisely and clearly these instructions are written, the faster, more organized, and more reliable the collaboration between human and AI will be on the project.

In the next chapter, we'll get to know another important topic: managing files in a Cowork project, and how Claude can interact effectively with the project's various files.

---

# Chapter 21: Managing Files in a Project

In the previous chapter, we got to know Global Instructions. Now it's time to learn how Claude interacts with a project's files and how we can optimize that interaction.

Managing files correctly is one of the key skills for working with Cowork.

## Why File Management Matters

In real projects, you're usually dealing with dozens or even hundreds of files. For Claude to work effectively on a project, it needs to be able to:

- Find relevant files
- Analyze their content
- Apply the necessary changes
- Create new files in the right location

If file structure and management aren't handled well, problems like these come up:

- Changes made to the wrong file
- Code created in the wrong location
- Dependencies getting overlooked
- Inconsistencies introduced into the project

For this reason, one of the important skills in using Cowork is properly managing the project's files.

## How Claude Interacts with Files

In a Cowork project, Claude can perform several main types of operations on files.

**1. Reading a file**

Claude can review a file's contents to understand its structure or logic.

Example:

*"Review the file* `src/services/user-service.ts` *"*

After reading the file, Claude can:

- Analyze the code's logic
- Identify bugs
- Offer improvement suggestions

**2. Editing a file**

One of the most common uses is editing existing files.

Example:

*"In the file* `src/controllers/user-controller.ts` *, add an endpoint for deleting a user."*

In this case, Claude:

1. Reads the file

2. Analyzes its structure
3. Applies the necessary changes
4. Puts the new code in the appropriate place

**3. Creating a new file**

Claude can also create new files.

Example:

*"Create a new service for managing orders. Path: `src/services/order-service.ts`"*

In this case, Claude:

- Creates the new file
- Writes the code structure according to the project's standards
- Adds the necessary imports

**4. Searching the project**

Claude can search across the entire project.

Example:

*"Find every place where the `getUserById` function is used."*

Or:

*"Show me all the files that depend on `userService`."*

This capability is very useful for understanding dependencies and analyzing code.

## Principles of File Naming

Proper file naming helps Claude understand the project structure faster.

**Good naming:**

```
user-service.ts
auth-controller.ts
product-model.ts
payment-middleware.ts
```

In this case, just by looking at the file name, you can tell:

- What the file does
- Which part of the project it belongs to

**Poor naming:**

```
service1.ts
main.ts
file.ts
utils.ts
```

Names like these don't provide useful information and create confusion in the project.

Simple rule: a file's name should indicate its role.

## Organizing Files

For better project management, files should be logically grouped.

**Feature-based structure**

One common approach is to organize by feature:

```
src/
├── users/
│   ├── user.controller.ts
│   ├── user.service.ts
│   ├── user.model.ts
│   └── user.validation.ts
├── products/
│   ├── product.controller.ts
│   ├── product.service.ts
│   └── product.model.ts
├── orders/
│   ├── order.controller.ts
│   ├── order.service.ts
│   └── order.model.ts
```

In this model, all files related to a single feature live in one folder.

**Benefits of this approach:**

- Faster understanding of the project
- Less file scattering
- Easier for Claude to work with different parts of the system

## Adding a New Feature

Suppose you want to add a comment-management feature to the project. In Cowork, you can make a request like this:

*"Create a complete module for managing comments:*
*- model:* `src/models/comment.model.ts`
*- service:* `src/services/comment-service.ts`
*- controller:* `src/controllers/comment-controller.ts`
*- routes:* `src/routes/comment-routes.ts` *"*

Claude creates the necessary files and writes the base code for each part.

After that, you can request:

*"Add the comment route to the* `app.ts` *file."*

Claude edits the main application file and registers the new route.

Key takeaway: by specifying the file paths precisely, Claude knows exactly where to put the code.

## Managing Dependencies Between Files

In large projects, files are usually dependent on one another. For example:

- A controller depends on a service
- A service depends on a model
- A middleware might be used across several routes

Claude can analyze these dependencies.

Example:

*"Check the dependencies of the* `user-service.ts` *file."*

Or:

*"If this file is deleted, which parts of the project would be affected?"*

This capability is very useful when refactoring or changing the project's architecture.

## Working with Large Files

Sometimes a file gets very large and becomes hard to manage. For example:

```
user-service.ts (1200 lines)
```

In cases like this, it's best to split the file into several parts. For example:

```
user-auth-service.ts
user-profile-service.ts
user-permission-service.ts
```

**Benefits of doing this:**

- Better code readability
- Greater testability
- Faster understanding for Claude

General rule: if a file goes beyond around 300-400 lines, you should probably split it.

## Best Practices for Working with Files in Cowork

For Claude to perform at its best, a few important recommendations:

**1. Specify the file path precisely**

Always write the file's complete path.

Weak: *"Edit the user service file."*

Better: *"Edit the file `src/services/user-service.ts`."*

**2. State the changes precisely**

Instead of a vague request, state exactly what change is needed.

Weak: *"Make this file better."*

Better:

*"In this file:*
*- Make the `getUserById` function async*

*- Add error handling*
*- Add input validation"*

**3. Use smaller files**

Smaller files are easier to analyze, and the chance of error is lower.

**4. Keep the project structure stable**

Constantly changing the structure creates confusion in the project. Define a clear structure and stick to it.

## Summary

File management is one of the key elements of working with Cowork projects. The clearer and more organized the file structure, the faster Claude can understand the project and apply more precise changes.

**Key takeaways from this chapter:**

- Proper file naming for clarity
- Logical organization based on features
- Managing dependencies between files
- Splitting large files into smaller parts
- Precisely specifying the path and the intended changes

Using these principles, you can build projects where human-AI collaboration is fast, transparent, and reliable.

In the next chapter, we'll get to know another important topic: connecting tools (Connectors), and how Claude can connect to external services and tools.

---

# Chapter 22: Connecting Tools (Connectors)

In the previous chapter, we learned how to manage a project's files. Now it's time to get to know one of Cowork's most powerful capabilities: connecting to external tools and services.

This capability turns Claude from a file-analysis tool into an active agent within the project's ecosystem.

## What Is a Connector?

One of Cowork's powerful capabilities is the ability to connect to external tools and services. We call these connections Connectors.

A Connector lets Claude:

- Connect to a database
- Communicate with external APIs
- Read or write files from cloud services
- Interact with project management tools
- Fetch data from various sources

In plain terms, a Connector is the bridge between Claude and the outside world.

## Why Do We Need Connectors?

Without a Connector, Claude can only work with files inside the project. But in the real world:

- Data is stored in databases
- Information comes from different APIs
- Files live in Google Drive or Dropbox
- Tasks are managed in Jira or Trello

By using a Connector, Claude can:

*"Get the list of active users from the database."*

Or:

*"Check the project's latest GitHub issues."*

Or:

*"Read the sales report file from Google Drive."*

This means Claude is no longer limited to local files and can interact with the whole project ecosystem.

## Common Types of Connectors

**1. Database connections**

One of the most common uses is connecting to a database.

Example: PostgreSQL

```
# .cowork/connectors/database.yml
type: postgresql
host: localhost
port: 5432
database: myapp_db
user: admin
password: ${DB_PASSWORD}
```

Once set up, you can ask Claude:

*"Get the number of users who signed up in the past month."*

Claude can:

- Write the appropriate query
- Run it
- Analyze the result

**2. Connecting to external APIs**

Claude can communicate with different APIs.

Example: the GitHub API

```
# .cowork/connectors/github.yml
type: github
token: ${GITHUB_TOKEN}
repository: username/project-name
```

Now you can say:

*"Show me the list of open pull requests."*

Or:

*"Create a new issue for the bug that was found."*

**3. Connecting to cloud services**

Claude can connect to cloud storage.

Example: Google Drive

```
# .cowork/connectors/gdrive.yml
type: google_drive
credentials: ${GDRIVE_CREDENTIALS}
folder_id: 1a2b3c4d5e6f
```

Then:

*"Read and analyze the 'Q1 sales report' file from Drive."*

**4. Connecting to project management tools**

Example: Jira

```
# .cowork/connectors/jira.yml
type: jira
url: https://company.atlassian.net
email: user@company.com
api_token: ${JIRA_TOKEN}
project_key: PROJ
```

Now you can say:

*"List this week's In Progress tasks."*

Or:

*"Create a new task for implementing the products API."*

## How to Set Up a Connector

Setting up a Connector usually involves a few steps.

**Step 1: Create the configuration file**

The Connector file lives in the `.cowork/connectors` folder.

Example:

```
.cowork/
└── connectors/
    ├── database.yml
    ├── github.yml
    └── slack.yml
```

**Step 2: Define the connection information**

Each Connector needs specific information.

Example: connecting to Slack

```
type: slack
workspace: mycompany
bot_token: ${SLACK_BOT_TOKEN}
channel: #dev-team
```

**Step 3: Use environment variables**

For security, use environment variables:

```
# .env
DB_PASSWORD=secret123
GITHUB_TOKEN=ghp_xxxxxxxxxxxx
SLACK_BOT_TOKEN=xoxb-xxxxxxxxxxxx
```

This prevents sensitive information from ending up in the code.

Security note: never commit the `.env` file to Git. Add it to `.gitignore`.

**Step 4: Test the connection**

After setup, test the connection:

*"Check the database connection."*

Or:

*"Send a test message to the Slack channel."*

## Practical Example: Connecting to a Database and GitHub

Suppose you have a project where:

- Data is stored in PostgreSQL
- The code is managed in GitHub

**Setting up the database Connector**

```
# .cowork/connectors/db.yml
type: postgresql
host: localhost
port: 5432
database: ecommerce_db
user: admin
password: ${DB_PASSWORD}
```

**Setting up the GitHub Connector**

```
# .cowork/connectors/github.yml
type: github
token: ${GITHUB_TOKEN}
repository: company/ecommerce-api
```

**Using them in practice**

Now you can make combined requests:

*"Get the list of products with less than 10 units in stock from the database, and create a GitHub issue for each one."*

Claude:

1. Connects to the database
2. Runs the appropriate query
3. Analyzes the results
4. Creates a GitHub issue for each product

This kind of automation can save a team hours of work.

## Security When Using Connectors

Connecting to external services means access to real data. So respecting security is very important.

**Principle 1: Use the least amount of privilege necessary**

The token or user account used for the connection should only have the minimum access it needs. For example:

- If you only need to read issues, don't grant write access
- If you're only pulling reports, don't grant delete access

**Principle 2: Don't store sensitive information in the code repository**

Never commit the following directly into a file:

- API tokens
- Database passwords
- Private keys

Always use environment variables:

```
DB_PASSWORD=********
GITHUB_TOKEN=********
```

**Principle 3: Log activities**

It's best to log sensitive activities:

- Running important queries
- Creating or deleting issues
- Sending messages to team tools

This creates transparency and traceability.

## Designing Smart Workflows with Connectors

The true power of Connectors becomes apparent when you combine them into a Workflow.

**Example 1: Error monitoring**

*"If the number of errors logged in the database today is more than 100, alert the team in Slack."*

Claude can:

1. Run the query
2. Check the condition
3. Send a message if needed

**Example 2: Project management automation**

*"Get the pull requests merged this week, and for each one, change the status of the corresponding Jira task to Done."*

This kind of automation saves the team a lot of time.

**Example 3: Automatic reporting**

*"Get this month's sales figures from the database, generate an analytical summary, and save the report file to Google Drive."*

In this case, Claude:

- Fetches the data
- Analyzes it
- Produces the output
- Uploads the file

Key takeaway: these kinds of Workflows can be run periodically (daily, weekly).

## Limitations and Important Notes

Despite the high power on offer, a few things need to be kept in mind:

- Always carefully review the request before running sensitive operations
- Get explicit confirmation for destructive operations (deletion, large-scale changes)
- Only enable Connectors for services that are actually necessary

It's also best to design complex Workflows step by step to prevent errors.

Simple rule: the more sensitive the operation, the more oversight it needs.

## Summary

Connectors turn Claude from a file-analysis tool into an active agent within the project's ecosystem. By connecting to:

- Databases
- APIs

- Project management tools
- Cloud services

many repetitive tasks can be automated, increasing the team's productivity.

**Key takeaways from this chapter:**

- A Connector is the bridge between Claude and external services
- Security is the first priority when setting up Connectors
- Combining several Connectors can create powerful Workflows
- Always start with the least amount of access necessary

In the next chapter, we'll talk about a key topic: designing workflows, and how to turn human-Claude interaction into a structured, professional process.

---

# Chapter 23: Designing Workflows

## 1. What Is a Workflow?

A Workflow is an organized, repeatable flow of work designed to accomplish a task or reach a goal.

In working with Claude, a Workflow means:

- Defining clear steps for accomplishing a task
- Dividing responsibilities between the human and Claude
- Creating a repeatable, optimized process

Instead of talking to Claude freeform every single time, you design a fixed working pattern that increases efficiency.

## 2. Why Do We Need Workflows?

**Without a Workflow:**

- You have to start from scratch every time
- Important steps might get forgotten
- The quality of the output varies
- Team collaboration becomes difficult

**With a Workflow:**

- The process is standardized and repeatable
- Quality is consistent and predictable
- Team members know what they need to do
- Productivity increases

## 3. Components of a Good Workflow

An effective Workflow includes these parts:

**a) Input**

What information is needed to start the task?

Example:

- A description of the new feature
- Related files
- Technical requirements

**b) Steps**

What stages is the work broken into?

Example:

1. Requirement analysis
2. Architecture design
3. Implementation
4. Testing
5. Documentation

**c) Responsibilities**

At each step, who does what?

- Human: decision-making, review, approval
- Claude: analysis, code generation, suggesting solutions

**d) Output**

What's the final result?

Example:

- Tested code
- Complete documentation
- An analysis report

**e) Quality Criteria**

How do we know the work was done correctly?

Example:

- All tests pass
- The code follows the project's standards
- Documentation is complete

## 4. Practical Example: A Workflow for Adding a New Feature

Suppose you want to add a new feature to the project. Let's design this Workflow:

**Step 1: Requirement analysis**

Human:

*"I want to add a feature to filter products by category."*

Claude:

1. Asks follow-up questions
2. Specifies the technical requirements
3. Identifies edge cases

Output: a requirements analysis document

**Step 2: Solution design**

Human:

*"Based on the analysis, propose a technical solution."*

Claude:

- Reviews the architecture
- Specifies the necessary changes to the files
- Designs the new API endpoint
- Suggests database changes (if needed)

Output: a technical plan including the files and changes

**Step 3: Review and approve the plan**

Human:

- Reviews the plan
- Raises questions or requests changes
- Approves the final plan

**Step 4: Implementation**

Claude:

- Generates the code based on the plan
- Follows the project's standards
- Adds the necessary comments

Output: the implemented code

**Step 5: Generating tests**

Claude:

- Writes unit tests
- Writes integration tests
- Covers various scenarios

Output: test files

**Step 6: Running the tests**

Human:

*"Run the tests."*

Claude:

- Runs the test command
- Reports the results
- Fixes issues if any errors occur

**Step 7: Documentation**

Claude:

- Adds docstrings
- Updates the README
- Writes a usage example

**Step 8: Final review**

Human:

- Reviews the code
- Checks the tests
- Requests minor changes if needed

Final output: a feature ready to merge

## 5. Workflows for Different Kinds of Tasks

**a) Bug-fix Workflow**

1. Describe the bug and reproduce it
2. Analyze the root cause
3. Suggest a solution
4. Implement the fix
5. Run regression tests
6. Document it

**b) Refactoring Workflow**

1. Identify problematic code
2. Analyze the issues
3. Design a better structure
4. Refactor incrementally
5. Make sure behavior doesn't change
6. Update the tests

**c) Code Review Workflow**

1. Claude reads the code
2. Checks against standards
3. Identifies potential issues
4. Suggests improvements

5. Reviews security and performance
6. Produces a review report

**d) Research and Learning Workflow**

1. Define the question or topic
2. Search the codebase
3. Analyze patterns
4. Summarize the findings
5. Produce educational documentation

## 6. Recording a Workflow in the Project

For Workflows to be reusable, document them.

Example: a Workflow file

```
# .cowork/workflows/add-feature.md

## Workflow: Adding a New Feature

### Input
- Feature description
- Technical requirements
- Related files

### Steps
1. Requirement analysis
2. Technical design
3. Plan approval
4. Implementation
5. Writing tests
6. Running tests
7. Documentation
8. Final review

### Acceptance criteria
- All tests pass
- Global Instructions standards are followed
- Documentation is up to date
```

This way, every team member can run the same standardized process.

## 7. Designing Step-by-Step Workflow Execution

One of the best practices is to ask Claude to execute step by step and wait for approval after each step.

Example:

*"Run this Workflow step by step. Stop after each step and wait for approval."*

**Benefits:**

- More control over changes
- Fewer large errors

- Ability to correct course early

## 8. Combining Workflows with Connectors

Workflows become even more powerful when combined with Connectors.

Example: releasing a new version

1. Check the tests
2. Bump the version in `package.json`
3. Create a tag in GitHub
4. Build a release
5. Send a notification message in Slack

Claude can:

- Run the tests
- Change the version
- Create the tag
- Send the message

All within one organized workflow.

## 9. Optimizing Workflows

After running a Workflow a few times, review it:

- Which step is repetitive?
- Where do errors happen most often?
- Which step should be automated?
- Which step needs human approval?

A good Workflow is stable, but still improvable.

## 10. Golden Principles for Designing Workflows

1. Start simple — add complexity incrementally
2. Define steps clearly — ambiguity causes errors
3. Specify responsibilities — who does what
4. Set acceptance criteria — how we measure success
5. Require approval for sensitive operations — security comes first

6. Document and version your Workflow — preserve the experience

## Summary

A working Workflow turns your interaction with Claude from a simple conversation into a professional, structured system.

By designing a Workflow:

- Output quality becomes stable
- Team collaboration becomes simpler
- Errors are reduced
- Automation increases

**Key takeaways from this chapter:**

- A Workflow is an organized, repeatable flow of work
- Every Workflow includes input, steps, responsibilities, output, and quality criteria
- Step-by-step execution gives you more control
- Combining Workflows with Connectors multiplies the power of automation
- Workflows should be documented, repeatable, and improvable

This now completes Part Four of the book (Cowork and Project Management). In the next part, we'll move into a different but very practical area: using Claude for content creation and professional writing.

---

# Part Five: Claude for Content Creation and Writing

# Chapter 24: Defining Your Writer's Voice

## 1. What Is a Writer's Voice?

A writer's voice, or brand voice, refers to the consistent personality and tone that shows up across all of a writer's or a brand's content. This voice determines how you speak to your audience and what feeling you convey to them.

When a writer has a distinct voice, the audience gradually gets to know their style. They might even be able to guess who wrote a piece of text without seeing the author's name.

**The main elements of a writer's voice**

A writer's voice is usually made up of several elements:

- Writing tone: formal, warm, serious, or humorous
- Type of vocabulary: simple, technical, creative
- Sentence structure: short and punchy, or long and explanatory
- Degree of formality or warmth: using formal or informal address
- The way it views the audience: as a guide, a friend, a teacher, or an advisor

For example, some brands speak in a completely formal, professional way, while others are very warm and friendly. Both can be right — what matters is that this style stays consistent across all the content.

**Example: Two different voices**

Brand A (a tech startup):

*"We're here to make your job easier!*
*With our tools, there's no more need for manual work.*
*Come try it out."*

Brand B (a management consulting firm):

*"We provide strategic solutions to improve organizational performance.*
*Achieve your goals through precise analysis and professional planning."*

The difference is clear: one is warm and cheerful, the other formal and professional.

## 2. Why Does a Consistent Voice Matter?

Consistency in tone is one of the most important factors in building a content identity.

**Without a consistent voice:**

- Content feels inconsistent
- The audience can't recognize you
- Less trust is built
- The brand lacks a clear personality

**With a consistent voice:**

- **Recognizability increases:** the audience remembers you faster
- **Trust increases:** consistency in tone makes the brand seem more professional and trustworthy

- **An emotional connection forms:** the audience feels like they're talking to a real personality
- **Content becomes more unified:** even if several people produce the content, they all share one voice

For this reason, many big brands define a tone guide or style guide for their content.

## 3. Teaching Claude Your Tone

Language models get the best results when they know exactly what style to write in. There are three common ways to teach Claude a tone.

**Method one: Providing sample text**

In this method, you give Claude a few samples of writing that show your desired style.

**Sample prompt:**

*"These are a few samples of my writing tone. Please review their style, then write a new piece of text in the same tone."*

*Sample texts:*

*Text one:*
*"A lot of people think success is a sudden event. But most success is the result of years of effort and mistakes."*

*Text two:*
*"If you want to start something, don't wait for perfect conditions. Most of the time, those conditions don't exist at all."*

*Text three:*
*"Sometimes the most important step is just to start, even if the path isn't entirely clear yet."*

When Claude sees these samples, it identifies patterns like simple language, short sentences, and a motivational tone, and produces new text in the same style.

**Method two: Directly defining tone characteristics**

Sometimes, instead of a sample text, you can directly describe the tone's characteristics.

Example:

*"The text should be written with these characteristics:*
*- Warm and friendly*

*- Short sentences*
*- Simple, understandable language*
*- A bit motivational*
*- No complicated jargon*

*Now write a post about the importance of planning."*

This method is useful when you know exactly what qualities you want, but don't have a suitable sample text.

**Method three: Specifying constraints**

In many cases, saying what shouldn't be in the text is also very useful.

Example:

*"Don't use these in the text:*
*- A formal, bureaucratic tone*
*- Very long sentences*
*- Complicated technical jargon*
*- Exaggerated claims*
*- Heavy use of emojis"*

This makes the output land much closer to the intended style.

**Practical example: Claude's output**

Prompt:

*"Based on a warm, simple tone, write a post about the importance of rest."*

Output:

*"Sometimes the best thing you can do is nothing at all.*
*Yeah, you read that right. Resting is a skill too.*
*When you keep working nonstop and never give yourself a break, your brain gets tired.*
*Your creativity drops. Your motivation fades.*
*So set aside one day a week just for yourself. Read a book, go for a walk, or just sleep.*
*Your body will thank you. So will your mind."*

The tone is completely warm, simple, and motivational.

## 4. Building a Writing Profile

If you're going to produce content with Claude on an ongoing basis, it's best to build a writing profile. This profile is essentially a short document that defines your writing style.

**Sample of a simple profile**

```
# Writing Profile

## Target audience
Entrepreneurs, freelancers, and people interested in learning new
skills.

## Content tone
Warm, simple, and motivational.

## Sentence style
Short, clear sentences. Preferably under 20 words.

## Type of vocabulary
Everyday and easy to understand. No complicated academic jargon.

## Content characteristics
- An engaging opening with a question or a thought-provoking sentence
- Uses an example or personal experience
- Ends with one practical takeaway

## Things to avoid
- A dry, formal tone
- Excessive exaggeration
- Long, complicated sentences
- Overuse of emojis

## Sample content
[real samples of previous content]
```

When this profile is given to Claude, the model can keep the same style across all subsequent content. This way, the output becomes more consistent and needs less editing.

**Using a profile in Cowork**

If you're using the Cowork environment, you can put this profile in the `.cowork/instructions.md` file:

```
# Global Instructions

## Writer's Voice
All generated content must be written in this tone:
- Warm and friendly
- Short sentences (maximum 20 words)
- Simple, everyday language
- Motivational but realistic
- No complicated jargon

## Sample content
[real samples]
```

With this, Claude respects this tone across all interactions with the project.

## 5. The Different Dimensions of a Writer's Voice

A complete writer's voice includes several dimensions:

**a) Tone**

- Formal: suitable for large corporations, legal, financial
- Warm: suitable for personal brands, startups
- Professional but friendly: a balance between the two above

**b) Personality**

- Serious and thoughtful: focused on analysis and depth
- Cheerful and energetic: uses humor and excitement
- Calm and supportive: emphasizes empathy and guidance

**c) Vocabulary**

- Technical: uses specialized terminology
- Simple: everyday, understandable language
- Creative: uses metaphor and imagery

**d) Sentence structure**

- Short sentences: fast and direct
- Long sentences: explanatory and analytical
- A mix: variety for engagement

## Summary

One of the most important steps in producing content with language models is precisely defining the writer's voice. If the tone, audience, and writing style are properly specified, Claude can produce text that's very close to the actual style of the writer or brand.

**Key takeaways from this chapter:**

- A writer's voice consists of tone, vocabulary, sentence structure, and personality
- Consistency in voice creates recognizability, trust, and emotional connection
- Three main methods for teaching tone: providing samples, direct definition, and specifying constraints
- Building a writing profile creates consistency across content
- In Cowork, you can put the profile inside Global Instructions

In the next chapter, we'll cover the practical application of this topic in generating social media content.

---

# Chapter 25: Generating Social Media Content

## 1. Building a Post Template

Producing content for social media on an ongoing basis, without a clear structure, is usually hard and time-consuming. Many content creators start from scratch every single time, and that slows things down and creates inconsistency in tone and quality across posts.

One of the best ways to solve this problem is to build a post template.

**What is a post template?**

A post template is a predefined structure that specifies how each post should start, what sections it should have, and how it should end. Once the template is set, all you have to do is change the topic, and Claude can generate the complete text.

**Sample template for an educational post**

**Part one: An engaging opening**

A question, an interesting statistic, or a sentence that grabs the audience's attention.

**Part two: Explaining the problem**

Explain the problem or main topic in two or three sentences.

**Part three: Offering a tip or solution**

Offer a few practical tips or a simple solution.

**Part four: A summary**

Sum up the post's main message in one sentence.

**Part five: A call to interact**

Ask the audience to write their opinion or share their experience.

**Sample prompt for Claude**

*"Based on the template above, write an Instagram post about time management for freelancers. The tone should be warm and simple."*

**Sample output:**

*"Ever get to the end of the day feeling like you had a ton of work, but nothing actually moved forward?*

*This is one of the biggest challenges freelancers face.*

*When you're your own boss, managing time is harder than it looks.*

*Here's a simple approach: pick just three main tasks each day. Not ten, not fifteen. Just three important tasks.*

*When you focus on just those three, you're much more likely to actually finish them.*

*Now tell me: what's your biggest challenge with managing your time?"*

Having a few different templates makes content production much faster. Content creators usually have a few fixed templates — like an educational template, a story template, a list template, and a Q&A template.

**Common types of templates**

**1. Educational template**

- Start with a question or problem
- A brief explanation
- Offer a solution
- Invite interaction

**2. Story template**

- Start with a personal experience
- Explain the challenge
- The lesson learned
- The final message

**3. List template**

- Introduce the topic
- List of tips (3 to 7 items)
- A short wrap-up
- Invite interaction

**4. Q&A template**

- Pose a common question
- A brief, practical answer
- Additional explanation
- Invite more questions

## 2. Generating a Content Calendar

One of the most important factors for success on social media is publishing content regularly. If posting is irregular, page growth slows down considerably.

A content calendar is a tool that helps you plan your posting schedule ahead of time.

**Elements of a content calendar**

A content calendar usually specifies:

- Post topic: what it's about
- Content type: educational, motivational, story-based, or interactive
- Publish date: which day it goes out
- Post goal: increasing engagement, education, or introducing a product

Claude can design a complete content calendar within seconds.

**Sample prompt**

*"Design a 30-day content calendar for an Instagram page in the personal-development niche. 5 posts are published each week. Content types should include educational, motivational, personal experience, and Q&A."*

**Sample output:**

**Week one**

Monday: Educational post
Topic: Three common time-management mistakes

Tuesday: Motivational post
Topic: Why starting matters more than being perfect

Thursday: Personal experience
Topic: My first work failure and the lesson I took from it

Friday: Q&A
Topic: How do we build a reading habit?

Saturday: List post
Topic: Five morning habits of successful people

**Week two**

Monday: Educational post
Topic: How to set actionable goals

Tuesday: Motivational post
Topic: When you feel like you're not making progress

Thursday: Personal experience
Topic: The biggest mistake in starting a business

Friday: Q&A
Topic: How do we improve our focus?

Saturday: List post
Topic: Four simple tools for daily planning

With a calendar like this, you know exactly what content to produce each day. This makes content production more organized, faster, and more purposeful.

**Benefits of a content calendar**

- **Long-term planning:** an overview of a month's or three months' worth of content
- **Content variety:** ensures different types of posts exist
- **Reduced stress:** no need to think daily about what to post
- **Consistency in publishing:** maintaining a regular presence on social media

## 3. A Real Example of a Campaign

One important use of Claude on social media is designing content campaigns. A campaign is a series of related posts designed to reach a specific goal. That goal might be introducing a product, attracting a new audience, or increasing engagement.

**Example: Launching an online course campaign**

Suppose you want to introduce an online course about time management. You could design a one-week campaign.

**Sample prompt**

*"To introduce an online time-management course, design a 7-day campaign for Instagram. The campaign's goal is to increase course sign-ups."*

**Sample campaign structure**

**Day one — Awareness**

Post: The common problem of poor time management
Goal: help the audience realize this problem is common to many people.

**Day two — A quick lesson**

Post: A simple time-management technique
Goal: provide real value to the audience.

**Day three — A story**

Post: A personal experience from when you didn't manage your time well
Goal: create an emotional connection with the audience.

**Day four — Common mistakes**

Post: Five common planning mistakes
Goal: demonstrate knowledge and expertise.

**Day five — Introducing the solution**

Post: introduce the course and explain what it teaches
Goal: present the product as the solution.

**Day six — Answering questions**

Post: answer common questions about the course
Goal: resolve doubts and increase trust.

**Day seven — A final call**

Post: a reminder to sign up and an announcement of the last chance
Goal: create a sense of urgency and encourage action.

**Sample text for one of the campaign posts (day five)**

*"A lot of people think their problem is a lack of time.*

*But often the real problem is a lack of a time-management system.*

*If every day starts without a plan, it's natural for important tasks to fall behind. That's exactly why, in the time-management course, we learn how to design our day with a clear purpose.*

*If you want to take back control of your time, the sign-up link is in the bio."*

A campaign like this helps the audience get to know your problem, your solution, and eventually your product, step by step.

**Why are content campaigns effective?**

- **Gradual awareness building:** the audience gets familiar with the topic gradually
- **Increased trust:** offering valuable content before selling
- **Creating a sense of urgency:** setting a deadline for action
- **Increased engagement:** varied posts drive more interaction

## 4. Key Points in Producing Social Media Content

**a) Know your audience**

Before producing content, you need to know who your audience is:

- What age are they?

- What are their interests?
- What problems do they have?
- What type of content do they enjoy?

**b) Content variety**

If you always produce one type of content, the audience gets bored. Use a mix of these:

- Educational posts
- Motivational posts
- Personal stories
- Q&A
- Entertaining content

**c) Invite interaction**

At the end of every post, ask the audience to:

- Write their opinion
- Share their experience
- Ask a question
- Save or share the post

**d) Use appropriate hashtags**

Hashtags help content get seen. Use a mix of popular and niche hashtags.

**e) Analyze performance**

After publishing content, review its performance:

- Which posts got more engagement?
- What type of content does the audience like more?
- What's the best time to publish?

## Summary

Social media is one of the most important channels for connecting with your audience, but producing content for it continuously can be time-consuming. With Claude's help, this process can become much faster and more organized.

**Key takeaways from this chapter:**

- Building fixed templates for posts speeds up content production
- A content calendar helps with long-term planning and regular publishing
- Content campaigns are designed to reach specific goals
- Content variety and inviting interaction are success factors
- Claude can generate varied ideas and a complete calendar within seconds

Using this tool, you can have a more active presence on social media and build a better connection with your audience.

---

# Chapter 26: Generating Articles and Blog Posts

## 1. Designing an Article's Structure

Before you start writing, the most important step is designing the structure. Many writers start without a plan, and the result is a scattered article without a logical flow.

An article's structure is like a building's blueprint. If the blueprint is clear, building it will be much simpler. Claude can design a complete structure for an article within seconds.

**Sample prompt**

*"I want to write a 2000-word article about 'How to Build a Reading Habit.' Design the complete article structure, including the introduction, main sections, and conclusion."*

**Sample structure**

**Title:** How to Build a Reading Habit? A Practical Guide to Starting and Sticking With It

**Introduction (about 150 words)**

- Why do so many people fail at building a reading habit?
- The importance of reading in personal and professional growth
- The article's goal: to offer a practical method for building a lasting reading habit

**Section one: The main obstacles to building a reading habit (about 300 words)**

- Lack of time and digital distractions
- Starting with big, unrealistic goals
- Choosing unsuitable books to start with
- An unsuitable environment and lack of a clear routine

**Section two: The scientific principles of habit formation (about 300 words)**

- The role of repetition and consistency in cementing behavior
- The importance of starting small
- The relationship between environment, behavior, and habit
- Reinforcing the habit with small rewards and tracking progress

**Section three: Practical steps for building a reading habit (about 600 words)**

Step 1: Set a fixed, specific time for reading (e.g., 10 minutes a day)
Step 2: Choose a suitable, engaging book to start with
Step 3: Remove distractions and create a reading environment
Step 4: Track your progress and see the results
Step 5: Gradually increase your reading time

**Section four: Common mistakes on the path to building the habit (about 400 words)**

- Expecting fast results and getting discouraged
- Buying too many books without a plan
- Long gaps and struggling to restart
- Abandoning the habit due to lack of sufficient motivation

**Conclusion (about 250 words)**

- A summary of the key points (starting small, consistency, a suitable environment)
- Emphasizing that building a reading habit is a marathon, not a sprint
- A call to action: encouraging the reader to start a 10-minute reading session today

Having a structure like this makes writing the article much faster and more organized. You know exactly what content each section should contain.

## 2. Research and Gathering Information

Once the structure is set, the next step is gathering information. Even if you know the article's topic well, further research can raise the quality of the content.

Claude can help you at this stage with:

- Gathering initial, general information
- Suggesting related, deeper topics
- Designing key research questions
- Providing supplementary examples and explanations

**Sample prompt for research:**

*"For an article about building a reading habit, suggest important research questions I should find answers to before writing."*

**Sample output of research questions:**

- According to scientific research, how long does it typically take to form a new habit?
- What factors cause people to give up on a reading habit?
- Do digital and paper reading differ in terms of learning and information retention?
- What's the best time of day for reading, and why?
- What methods exist for maintaining reading motivation long-term?
- What kinds of books are more suitable for beginners, and how do we choose the right one?

These questions help you write a deeper, more precise, and more credible article.

## 3. Writing the Article Stage by Stage

One common mistake when using AI tools is asking them to write the entire article in a single step. The usual result of this is shallow, generic text.

A better approach is to write the article section by section. First, ask Claude to write the introduction. Then produce each section of the article separately.

This has several benefits:

- **The text quality goes up:** Claude can focus more on each individual section.
- **Editing each section is easier:** you can fix the content before moving on to the next section.
- **The text feels more natural:** the logical flow between sections is preserved better.

**Sample prompt for writing the first section:**

*"Based on the structure we designed, write the article's first section: 'Why is building a reading habit hard?' Tone: warm and instructional. Length: about 300 words. Include real examples."*

**Sample output (section one):**

*"A lot of people decide at the start of the year to read more. They put together long lists of recommended books and start off with high motivation. But after a few weeks, that motivation fades, and reading gets forgotten again.*

*Why does this happen?*

*One of the main reasons is unrealistic expectations. Many people think they need to read a book every week, or read for several hours a day. When that big goal isn't achieved right away, they feel like they've failed and give up on reading altogether.*

*The second reason is not having a fixed time. When you don't set aside a consistent time for reading in your daily schedule, other things (like checking social media or catching up on backlogged tasks) always take priority. Reading ends up at the bottom of the to-do list, and usually never happens.*

*The third reason is choosing the wrong books. Some people pick heavy, technical, or boring books to start with. When reading becomes difficult and exhausting, the initial motivation disappears quickly.*

*The fourth reason is an unsuitable environment. Reading in a noisy or cluttered space makes it hard to focus, and leaves you with an unpleasant reading experience."*

## 4. Editing and Improving the Article

The final step is editing the text. Even if the article was written with Claude's help, human editing is very important to make sure the tone, precision, and flow fully match your standards.

At this stage, you can use Claude for various tasks:

- Shortening long sentences: to improve readability
- Simplifying the text: for better audience comprehension
- Improving the text's flow: creating a smoother connection between sentences and paragraphs
- Adjusting the tone: to make sure it's consistent with your writing style
- Checking spelling and grammar

**Sample prompt for editing:**

*"Make this paragraph simpler and smoother, without changing its meaning."*

Or:

*"Edit this section of the article to make it more readable and engaging, with a warmer tone."*

Claude can rewrite the text and improve its quality.

## Summary

Writing quality articles requires structure, research, and editing. Claude can help the writer at every one of these stages.

By designing the article's structure, gathering information, writing section by section, and doing a final edit, you can produce articles that are organized, useful, and readable. Using Claude properly speeds up the writing process while preserving the quality of the content.

**Key takeaways from this chapter:**

- Designing the article's structure is the first and most important step
- Claude can help with initial research and suggesting research questions
- Writing the article stage by stage (section by section) improves the quality of the text
- Final editing by a human is essential for preserving tone and precision

---

# Chapter 27: Writing a Book with Claude

Writing a book is one of the most challenging writing projects there is. Many people have good ideas for a book, but they never actually start, or they stop partway through.

Claude can help you at every stage of writing a book: from turning an initial idea into a complete table of contents, to designing chapters and writing the content stage by stage.

## 1. Turning an Idea into a Book Outline

The first step to writing any book is turning a general idea into a defined structure. Many writers get stuck at this stage. They know what topic they want to write about, but they don't know how to organize the book.

With a few simple questions, Claude can design the book's complete table of contents.

**Sample prompt:**

*"I want to write a book about time management for entrepreneurs. Please design the book's complete table of contents, including the main parts and chapters."*

**Sample output:**

**Suggested title:** Time, the Entrepreneur's Real Capital

**Introduction**

- Why time management is different for entrepreneurs
- The structure of the book

**Part one: Understanding time in entrepreneurship**

- Chapter 1: Why do entrepreneurs never have enough time?
- Chapter 2: The difference between being busy and being effective
- Chapter 3: The real cost of wasted time

**Part two: The principles of time management**

- Chapter 4: Prioritization: How to identify important tasks
- Chapter 5: Delegation: What you shouldn't be doing yourself
- Chapter 6: Saying "no" to the wrong opportunities

**Part three: Tools and techniques**

- Chapter 7: The Time Blocking method
- Chapter 8: The Pomodoro technique for entrepreneurs
- Chapter 9: Using your calendar as a strategic tool

**Part four: Managing energy**

- Chapter 10: Why managing energy matters more than managing time
- Chapter 11: Identifying your most productive hours
- Chapter 12: Rest as a strategy

**Part five: Implementing it in the real world**

- Chapter 13: Designing an ideal day
- Chapter 14: Managing interruptions and emergencies
- Chapter 15: Maintaining work-life balance

**Conclusion**

- A summary of the key points
- A 30-day plan to get started

Having a table of contents like this makes writing the book much easier. You know exactly what you need to write in each chapter.

Important note: the initial table of contents isn't final. As you write the book, you can change chapters, rearrange them, or add new ones.

**Improving the outline:**

If you don't like the initial outline, you can ask Claude to improve it:

*"Review this table of contents and suggest what chapters should be added or what changes are needed."*

Or:

*"Suggest a different table of contents with a different approach. This time, focus more on practical examples."*

## 2. Designing the Chapters

Once the book's table of contents is set, the next step is designing the details of each chapter. Every chapter should have a clear structure:

- A chapter introduction: introducing the topic and its importance
- The main body: explaining concepts, providing examples and solutions
- A conclusion: summarizing the key points
- An exercise or action item: something the reader should do

Claude can design the exact structure of each chapter.

**Sample prompt:**

*"Chapter 4 of my book is about 'Prioritization: How to identify important tasks.' Design the complete structure of this chapter, including the main sections, examples, and a practical exercise."*

**Sample output:**

**Chapter 4: Prioritization — How to Identify Important Tasks**

**Introduction (300 words)**

- A short story: an entrepreneur who was always busy but never made progress
- The core problem: confusing urgent with important
- The chapter's goal: learning to identify what's genuinely important

**Section one: The difference between urgent and important (500 words)**

- Defining urgent tasks
- Defining important tasks
- The Eisenhower Matrix
- Example: sorting an entrepreneur's tasks into this matrix

**Section two: Three questions for identifying priorities (600 words)**

- Question one: does this task help my main goal?
- Question two: what happens if this task doesn't get done?
- Question three: can someone else do this task?
- A practical example for each question

**Section three: Common mistakes in prioritization (400 words)**

- Mistake one: doing easy tasks instead of important ones
- Mistake two: responding to every single request
- Mistake three: not having clear criteria for prioritizing

**Section four: A practical prioritization system (500 words)**

- The MIT (Most Important Tasks) method
- Choosing 3 important tasks per day
- Scheduling important tasks at the start of the day
- Example: an entrepreneur's daily schedule

**Conclusion (200 words)**

- A summary of the key points
- Reminder: prioritization is a skill that improves with practice

**Practical exercise**

- Write today's task list
- Place them in the Eisenhower Matrix
- Choose three important tasks for tomorrow

## 3. Writing Stage by Stage

Once the book's table of contents and each chapter's structure are set, the main stage begins: writing. Many people run into two challenges at this stage:

- They don't know where to start

- Or they try to write an entire chapter at once, all in one go

But the best approach to writing a book with Claude is stage-by-stage writing. That is, work through the book piece by piece. Each time, focus on just one small section.

**The standard approach to stage-by-stage writing:**

Break the work into 4 stages:

1. Write each chapter's introduction
2. Write the main sections
3. Write the conclusion
4. Review and integrate

At each stage, Claude can produce different versions so you can pick the best one.

**Sample stage-by-stage workflow:**

Suppose you want to write chapter 4.

**Stage one: Writing the introduction**

*"Write a 300-word introduction for chapter 4 on the topic of prioritization. Tone: instructional and smooth. Audience: entrepreneurs."*

Claude gives you an organized version. If you don't like it, you can say:

*"Write a second version that's simpler and shorter."* Or: *"Remove the short story at the beginning of the text."*

**Stage two: Writing the main sections**

Write each section separately: *"Write the 'difference between urgent and important' section. Include a real example."* Or: *"Write the common-mistakes-in-prioritization section with 3 examples. Keep the tone warm."*

**Stage three: Writing the conclusion**

Every chapter needs a short conclusion at the end: *"Write a 200-word conclusion to wrap up this chapter. Include one key takeaway and one practical recommendation."*

**Stage four: Integrating the chapter**

After all the sections are written, use Claude for the final wrap-up: *"Integrate all the written sections for chapter 4. Make the tone consistent and make the text's flow feel more natural."*

This stage helps make the final text cohesive and professional.

**Sample workflow for writing a complete chapter:**

For a 1500-2000 word chapter, it's enough to run through the following prompts step by step:

1. *"Give me the chapter structure."*
2. *"Write the introduction."*
3. *"Write the first section."*
4. *"Write the second section."*
5. *"Write the third section."*
6. *"Write the conclusion."*
7. *"Integrate all of this."*
8. *"Make the final version flow better."*

With this method:

- The text's quality stays high
- Structural errors are reduced
- The writing process becomes stable and fast

## Summary

Claude can make book-writing much simpler. In this chapter, we learned three main stages:

- Turning an idea into a book outline
- Precisely designing the chapters
- Stage-by-stage writing using versioning and review

By following this process, even a first-time writer can produce a professional, well-organized book.

**Key takeaways from this chapter:**

- The book's table of contents is your roadmap
- Designing each chapter's structure before writing is essential
- Stage-by-stage writing increases both quality and speed
- Final integration is essential for keeping a consistent tone

---

# Chapter 28: Professional Text Editing

Writing the first draft of a text is only half the journey of writing. The more important part is editing. Many texts that seem ordinary at first turn into highly professional writing with a good edit.

Editing helps you convey your message more clearly, more concisely, and with more impact. Claude can act as a powerful assistant in this process.

In this chapter, we'll cover three important uses of Claude in editing:

- Rewriting text
- Simplifying writing
- Improving tone and text flow

## 1. Rewriting Text

Sometimes you write a piece of text but feel the sentences are weak, or the message isn't coming across well. In this case, the best thing to do is rewrite it.

Rewriting means keeping the main idea, but expressing it in a better, more professional way. Claude can rewrite a sentence, a paragraph, or even an entire section.

**Sample prompt:**

*"Rewrite this paragraph to be clearer and smoother:*

*'Time management is one of the most important skills people need to learn because if they don't manage time properly, a lot of tasks won't get done, and as a result they'll have less success.'"*

**Sample output:**

*"Time management is one of the most important skills for success. When we can't manage our time properly, many important tasks go undone, and our progress slows down."*

Rewriting makes the text shorter, clearer, and more professional.

**Types of rewriting with a specific goal:**

You can ask Claude to rewrite with a particular purpose in mind:

**For more clarity:**

*"Rewrite this text to make it simpler and easier to understand."*

**For more impact:**

*"Write this text to be stronger and more impactful."*

**For shortening:**

*"Shorten this paragraph, but keep its meaning intact."*

**For changing the style:**

*"Rewrite this text to be more formal."*

Or:

*"Rewrite this text to be warmer."*

Important note: the more precise the instruction, the better the result. If you want the text written for a specific audience, specify that.

## 2. Simplifying Text

One of the most common writing problems is overcomplicating things. Many writers use long sentences and difficult words, when readers usually prefer simple, smooth text.

Simplifying means conveying the same idea in more understandable language.

**Sample complex text:**

*"In order to optimize organizational processes and enhance productivity levels, it is imperative that managers proceed by adopting novel approaches and leveraging modern tools."*

**Sample prompt:**

*"Make this text simpler and more understandable:*

*'In order to optimize organizational processes and enhance productivity levels, it is imperative that managers proceed by adopting novel approaches and leveraging modern tools.'"*

**Sample output:**

*"To improve the organization's performance, managers should use new methods and modern tools."*

The meaning hasn't changed, but the text has become much clearer.

**Useful prompts for simplifying:**

*"Make this text simpler and more understandable."*

*"Shorten the sentences in this paragraph."*

*"Replace complicated words with simple ones."*

*"Remove unnecessary words."*

**An important technique: converting passive sentences to active**

One important simplification technique is converting passive sentences to active ones.

**Before:**

*"This report was prepared by the analysis team."*

**After:**

*"The analysis team prepared this report."*

The second sentence is more direct and more natural.

**Sample prompt:**

*"Convert the passive sentences in this text to active."*

## 3. Improving Tone and Text Flow

A text's tone has a big impact on the reader's experience. A piece of text might be factually correct, but if it doesn't have the right tone, reading it becomes hard or tedious.

Claude can change a text's tone to match the audience.

**Common types of tone:**

**Formal tone**

- Suitable for work reports, academic papers, and official letters

**Semi-formal tone**

- Suitable for blog articles and educational content

**Warm tone**

- Suitable for social media and general content

**Example of a tone change:**

**Original text:**

*"Time management is a skill that can increase an individual's productivity."*

**Warm version:**

*"If you manage your time better, you'll see how much more you get done."*

**Formal version:**

*"Time management is an important skill for increasing individual productivity."*

**Sample prompts:**

*"Make the tone of this text warmer."*

*"Write this text more formally."*

*"Make the tone of this paragraph suitable for a general audience."*

**Improving text flow**

Sometimes the problem with a text isn't in the sentences themselves, but in the connection between them. If the transition between paragraphs isn't natural, the reader feels a sense of disconnect.

Claude can fix a text's flow.

**Sample prompt:**

*"Make this text's flow smoother and improve the connection between the paragraphs."*

Or:

*"Add a suitable transition sentence between these two paragraphs."*

This makes the text feel more unified and professional.

**Practical example:**

Suppose you have two paragraphs:

**Paragraph one:**

*"Time management is one of the most important skills for success. Without it, many tasks remain unfinished."*

**Paragraph two:**

*"The Pomodoro technique is one of the best time-management methods."*

**Prompt:**

*"Add a transition sentence between these two paragraphs."*

**Output:**

*"Time management is one of the most important skills for success. Without it, many tasks remain unfinished.*

*Fortunately, there are simple ways to improve your time management. The Pomodoro technique is one of the best time-management methods."*

The transition sentence lets the reader move easily from one topic to the next.

## Summary

Editing is one of the most important writing skills. Many professional texts are the result of several rounds of editing.

In this chapter, we learned three main uses of Claude in editing:

- Rewriting to improve text quality
- Simplifying to increase clarity
- Improving the tone and flow of writing

Using these methods, you can turn your writing from ordinary text into professional, readable prose.

**Key takeaways from this chapter:**

- Editing matters as much as writing
- Rewriting with a clear goal produces better results
- Simple writing is a strength, not a weakness
- Tone should match the audience
- A natural text flow improves the reading experience

---

# Part Six: Claude for Research and Academia

# Chapter 29: Literature Review

A literature review is one of the most important stages of any scientific research project. At this stage, the researcher examines what research others have already done on the topic, what results were found, and what gaps still exist.

The goal of a literature review isn't just to collect papers. Its main goal is to build a clear picture of the current state of knowledge in a field.

Claude can act as a powerful research assistant in this process. In this chapter, we'll cover three main uses:

- Searching for sources
- Summarizing papers
- Building a research literature map

## 1. Searching for Sources

The first step in a literature review is finding relevant scholarly sources. These sources usually include journal articles, conference papers, books, and research reports.

One of the main challenges researchers face is choosing the right keywords to search with. If the keywords aren't precise, you'll either find far too many sources, or miss relevant ones.

Claude can help design search keywords.

**Sample prompt:**

*"My research topic is 'The Impact of Social Media on Adolescent Mental Health.' What keywords would you suggest for searching for scholarly articles?"*

**Sample output:**

Claude will usually suggest a few kinds of keywords:

**Main keywords:**

- Social Media
- Mental Health
- Adolescents

**Combined keywords:**

- Social Media AND Mental Health

- Instagram AND Teenagers
- Online Platforms AND Psychological Well-being

**Related keywords:**

- Digital Media Use
- Cyber Psychology
- Online Behavior

These keywords can be searched in scholarly databases like:

- Google Scholar
- Scopus
- Web of Science
- IEEE Xplore
- ScienceDirect

**Initial screening of papers**

After finding papers, the next step is an initial screening of them. Usually, reading the title and abstract is enough to tell whether a paper is relevant.

Claude can help at this stage too.

**Sample prompt:**

*"Review this paper's abstract and tell me whether it's useful for research on the impact of social media on adolescent mental health.*

*[abstract text]"*

With this method, you can identify relevant papers much faster.

## 2. Summarizing Papers

After selecting the relevant papers, you need to read and summarize them. In many research projects, dozens of papers need to be reviewed, and this takes a lot of time.

Claude can summarize papers in a structured format.

**Sample prompt:**

*"Summarize this paper and extract the following four sections:*
*- The research goal*

*- The research method*
*- The main findings*
*- The conclusion*

*[the paper's text]"*

**Sample output:**

**Research goal:**

Examining the relationship between the level of social media use and anxiety levels in adolescents.

**Research method:**

A survey study of 300 high school students using a mental-health questionnaire.

**Findings:**

Using social media for more than three hours a day was linked to higher anxiety levels and lower life satisfaction.

**Conclusion:**

Heavy social media consumption may be linked to certain psychological issues in adolescents.

With this method, you can prepare a short, usable summary of every paper for your literature review.

**Comparing several papers**

Claude can also compare several papers together.

**Sample prompt:**

*"Compare these three papers and explain their similarities and differences in research method and results.*

*[Summary of paper 1]*
*[Summary of paper 2]*
*[Summary of paper 3]"*

This helps you identify common patterns or contradictory findings across the research.

## 3. Building a Research Literature Map

One of the most important goals of a literature review is understanding how previous research relates to one another. This overall structure is called a research literature map.

Claude can help you build this map. To do this, you can give Claude the summaries of several papers and ask it to categorize them.

**Sample prompt:**

*"Based on these paper summaries, build a research literature map and classify the studies into several thematic categories.*

*[paper summaries]"*

**Sample output:**

**Category one: The impact of the amount of social media use**

Studies examining the relationship between time spent on social media and mental health.

**Category two: The type of social media use**

Research examining the difference between active use (creating content) and passive use (viewing content).

**Category three: Mediating factors**

Studies examining the role of factors like self-esteem, social comparison, or social support.

With this categorization, you can design the structure of the literature review section of your paper or thesis.

**Identifying research gaps**

Claude can also help you identify research gaps.

**Sample prompt:**

*"Based on these studies, what topics haven't been sufficiently examined yet?*

*[paper summaries]"*

Answering this question is very important, because the research gap is exactly where your research can create new value.

**Sample output:**

*"Most studies have been done on American adolescents. There's little research on the impact of social media on adolescents in Middle Eastern countries.*

*Also, most research has focused on Instagram and Facebook, but the impact of newer platforms like TikTok has been examined less."*

This information helps you focus your research on where it's actually needed.

## Summary

A literature review is the foundation of every scientific research project. Without a precise understanding of prior research, designing new research is difficult.

In this chapter, we saw that Claude can help researchers with three important stages:

- Searching for sources and designing keywords
- Summarizing and comparing papers
- Building a research literature map

Using this tool properly, the literature review process becomes faster, more organized, and more thorough.

**Key takeaways from this chapter:**

- Precise keywords improve search quality
- Structured summarization of papers saves a lot of time
- A literature map helps you design the structure of your literature review
- Identifying research gaps is essential for defining a research question

---

# Chapter 30: Designing a Research Proposal

A proposal, or research plan, is the roadmap for any scientific research project. This document explains what question you want to answer, why that question matters, and how you intend to answer it.

Writing a proposal is one of the most challenging stages of research, especially for first-time students. Claude can be a huge help in this process.

In this chapter, we'll cover the three main stages of designing a proposal:

- Formulating the research question

- Writing the research method
- The proposal's complete structure

## 1. Formulating the Research Question

Every scientific research project starts with a question. But not every question is a good research question.

A good research question should be:

**Specific**

Not too broad, not too narrow.

**Investigable**

It should be possible to answer it with scientific methods.

**Important**

Answering it should have scientific or practical value.

Claude can help you refine your research question.

**Sample prompt:**

*"I want to research artificial intelligence in education. What's a good research question I could pose?"*

Claude usually offers a few suggested questions:

**A generic question:**

*"Is using artificial intelligence in education helpful?"*

This question is too broad and can't be precisely investigated.

**A better question:**

*"Does using intelligent educational systems improve the academic performance of math students?"*

This question is more specific and can be examined.

**A more advanced question:**

*"What factors moderate the impact of intelligent educational systems on students' math learning?"*

This question is deeper and seeks to understand mechanisms.

You can also give Claude your own question and ask it to improve it.

**Sample prompt:**

*"My research question is: 'Is artificial intelligence good?' Improve this question."*

Claude will usually explain why the current question isn't suitable and suggest a few better versions.

## 2. Writing the Research Method

The research method section explains how you plan to answer your research question. This section includes a few main parts:

**Research design**

Is your research experimental, survey-based, qualitative, or mixed-methods?

**Population and sample**

Who or what are you researching?

**Data collection tools**

How will you collect data? Questionnaire, interview, test?

**Data analysis method**

How will you analyze the data?

Claude can help design each of these sections.

**Sample prompt:**

*"My research question is: 'Does using intelligent educational systems improve the academic performance of math students?' What research method would you suggest?"*

Claude usually suggests a few options:

**Method one: An experimental study**

Select two groups of students. One group uses the intelligent system, and the other studies in the traditional way. Then compare their performance.

**Method two: A quasi-experimental study**

If you can't randomly assign students to groups, you could use existing classes.

**Method three: A survey study**

Ask students who've already used the intelligent system how their experience has been.

Claude can also provide more details.

**Sample prompt:**

*"What data collection tool would be suitable for this research?"*

The answer might include things like:

**An academic performance questionnaire**

Math test scores before and after using the system.

**A learning motivation questionnaire**

To measure the system's effect on student motivation.

**Semi-structured interviews**

To gain a deeper understanding of students' experience.

With this guidance, writing the research method section becomes much simpler.

## 3. The Proposal's Complete Structure

A proposal usually includes a few main sections:

**Introduction**

Introducing the topic and its importance.

**Literature review**

A summary of prior research.

**Research question**

The main question or research hypotheses.

**Research method**

An explanation of how the research will be conducted.

**References**

A list of papers and books used.

Claude can help you build the proposal's complete structure.

**Sample prompt:**

*"Build a complete structure for my research proposal. Topic: the impact of intelligent educational systems on math learning."*

The output usually looks like this:

**Research title**

The Impact of Intelligent Educational Systems on the Academic Performance of Math Students

**Introduction**

- The importance of learning math
- The challenges of traditional education
- The role of technology in education
- Introducing intelligent educational systems

**Literature review**

- Research on technology in education
- Studies on intelligent educational systems
- Research gaps

**Research question**

Does using intelligent educational systems improve the academic performance of math students?

**Hypotheses**

- Students who use the intelligent system will achieve better grades
- Their learning motivation will increase compared to the control group

**Research method**

- Research design: experimental with a control group
- Study population: first-year math students

- Sample: 60 students (30 per group)
- Data collection tools: a pre-test and post-test, a motivation questionnaire
- Data analysis method: an independent t-test to compare the means of the two groups

**References**

A list of papers and books related to the topic

## Advanced Use of Claude in Proposal Design

Beyond generating the structure, you can ask Claude to rewrite specific sections in a formal, academic style.

**Sample prompt:**

*"Rewrite this introduction paragraph to be more formal and suitable for a doctoral proposal."*

Or:

*"Make these hypotheses more precise and testable."*

You can also ask Claude to find logical flaws in your proposal.

**Sample prompt:**

*"This is my proposal text. Identify its potential weaknesses in terms of methodology."*

Claude can identify things like:

- An insufficient sample size
- A mismatch between the research question and the analysis method
- Key variables not being defined
- Ambiguity in the measurement tool

This kind of feedback is extremely valuable for improving the proposal's quality.

## Summary

Designing a proposal is a critical stage in any academic research project. If this stage is done precisely and logically, the rest of the research journey becomes much simpler.

In this chapter, we saw that Claude can help with three main sections:

- Formulating and improving the research question

- Designing and describing the research method
- Setting up the proposal's complete, formal structure

Using this tool intelligently, you can prepare a coherent, precise, and professional proposal that has a much better chance of being approved by your advisor or scientific committee.

**Key takeaways from this chapter:**

- A good research question should be specific, investigable, and important
- The research method should be consistent with the research question
- The proposal's structure should be logical and coherent
- Claude's feedback can identify methodological weaknesses

---

# Chapter 31: Analyzing Research Data

After data collection, the most important stage of research begins: analyzing the data. At this stage, raw data is turned into meaningful findings, and the researcher can answer their research question.

The data analysis method depends on the type of research. In different fields, there are usually two types of data analysis: qualitative analysis and quantitative analysis. Some research also uses a mixed-methods approach.

Claude can help at many stages of data analysis, from organizing qualitative data to interpreting statistical results.

In this chapter, we'll cover three main topics:

- Qualitative data analysis
- Quantitative data analysis
- Extracting research findings

## 1. Qualitative Data Analysis

Qualitative data is usually text-based — such as interviews, open-ended questionnaire responses, field notes, or documents.

The goal of qualitative analysis is to discover patterns, concepts, and hidden meanings in the data. This type of analysis focuses more on interpretation than calculation.

The qualitative data analysis process usually involves several stages.

**Stage one: Getting familiar with the data**

The researcher needs to read the interview transcripts or documents several times to get an overall understanding of the data.

**Stage two: Initial coding**

At this stage, important parts of the text are marked with labels called "codes." Each code represents an important concept in the data.

Claude can be very useful at this stage.

**Sample prompt:**

*"The following text is a portion of an interview with students about online learning. Extract the main codes."*

After reviewing the text, Claude might suggest codes like these:

- Internet problems
- Reduced interaction with the professor
- Flexibility in study time
- Increased independent learning
- Fatigue from screen time

**Stage three: Categorizing the codes**

After extracting the codes, you need to group them into broader categories.

**Sample prompt:**

*"Categorize these codes into a few main categories."*

The output might look like this:

**Category one: Technical challenges**

- Internet problems
- Poor quality of the online connection

**Category two: Learning experience**

- Increased independent learning
- Flexibility in the study schedule

**Category three: Educational interaction**

- Reduced interaction with the professor
- A lack of classroom discussions

**Stage four: Extracting themes**

At this stage, the researcher identifies the main patterns. These patterns are called "themes" and represent the research's main findings.

Claude can help identify the main themes.

**Sample prompt:**

*"Based on these categories, extract the main themes of the students' experience with online education."*

The output might include themes like these:

**Theme one: The duality of freedom and isolation**

Students are satisfied with the time flexibility, but experience a sense of isolation and reduced social interaction.

**Theme two: Infrastructure challenges**

Technical problems are the main barrier to effective online learning.

**Theme three: A shift in the learner's role**

Students have had to become more independent learners and manage their time better.

These themes can form the basis of the "qualitative findings" section in a paper or thesis.

## 2. Quantitative Data Analysis

Quantitative data consists of numerical information — such as test scores, results from scaled questionnaires, or statistical measurements.

The goal of quantitative analysis is to examine relationships between variables and test research hypotheses. This type of analysis usually uses statistical methods.

Claude can help with a few important parts:

- Choosing the appropriate statistical test
- Interpreting statistical results

- Writing a scientific report of the results

For example, suppose a researcher wants to compare the performance of two groups of students.

**Sample prompt:**

*"I have two groups of students. One group used a new educational tool, the other didn't. I want to compare their average grades. What statistical test is appropriate?"*

The answer will usually be:

- If the data is normally distributed, an independent t-test can be used.
- If the data isn't normal, the Mann-Whitney test is a more suitable option.
- If there are more than two groups, analysis of variance (ANOVA) can be used.

Claude can also be used to interpret results.

**Sample prompt:**

*"My test result is: t(58) = 2.91, p = 0.005. What does this result mean?"*

The interpretation might be something like this:

*"Since the p-value is less than 0.05, the difference between the two groups is statistically significant. Therefore, it can be concluded that the educational intervention likely had a real effect on student performance."*

Claude can also write statistical results in the formal style suitable for a scientific paper.

**Example:**

*"The results of the independent t-test showed that the mean score of the experimental group was significantly higher than the control group, t(58) = 2.91, p = 0.005."*

This kind of writing is exactly what's used in scientific papers.

## 3. Extracting Findings

After running the analyses, the researcher needs to extract and interpret the results. This stage involves answering the research question and explaining what the findings mean.

Many researchers run into trouble at this stage, because they don't know how to get from data to a conclusion.

Claude can help turn raw findings into logical conclusions.

**Sample prompt:**

*"My research findings show that using an intelligent educational system increased students' average grades. What conclusions can be drawn from this finding?"*

The answer is usually presented at a few levels.

**Level one: The direct result**

An intelligent educational system can help improve students' academic performance.

**Level two: An educational interpretation**

Features like immediate feedback, adaptive exercises, and greater interaction with content likely improved learning.

**Level three: Practical implications**

Using this kind of system could be worth considering in universities' educational programs.

You can also ask Claude to identify the limitations of the results.

**Sample prompt:**

*"Based on this research, what limitations might exist?"*

The answer might include things like:

- A limited sample size
- A focus on one specific field
- The short duration of the research

Stating these limitations makes the research come across as more realistic and credible.

Finally, the researcher can ask Claude to prepare a summary of the findings.

**Sample prompt:**

*"Based on these results, write a summary paragraph for the findings section of the research."*

This paragraph can be used directly at the end of the findings chapter of a thesis or paper.

## Summary

Data analysis is the stage where raw data gets transformed into scientific knowledge. In this chapter, we saw that Claude can help with three important areas:

- Analyzing qualitative data and coding texts
- Interpreting statistical results in quantitative analyses
- Extracting findings and drawing conclusions

Using this tool intelligently can make the data-analysis process faster, more precise, and more organized, and helps researchers present their findings in a scientific and presentable form.

**Key takeaways from this chapter:**

- Qualitative analysis involves coding, categorizing, and extracting themes
- Claude can suggest the appropriate statistical test
- Interpreting results should happen at different levels (direct, interpretive, practical)
- Stating limitations increases the research's credibility

---

# Chapter 32: Writing a Scientific Paper

After finishing the research and analyzing the data, another important stage begins: writing the scientific paper. A scientific paper is the main way knowledge gets transmitted in academia. Researchers publish their results through papers so others can use them, review them, or build further research on top of them.

Writing a scientific paper is different from writing ordinary text. This kind of writing needs to be precise, organized, well-documented, and based on standard scientific structures. Many researchers run into trouble turning their research into a coherent paper.

Claude can help at every stage of writing a paper: designing the structure, writing the different sections, adjusting the scientific tone, and final editing.

In this chapter, we'll cover three main topics:

- The structure of a scientific paper
- Writing the paper's different sections
- Editing and preparing it for submission

## 1. The Structure of a Scientific Paper

Most scientific papers follow a standard structure called IMRaD:

- Introduction

- Methods
- Results
- Discussion

Beyond these sections, a paper usually also includes:

- The title
- The abstract
- Keywords
- References

Claude can help you design your paper's structure.

**Sample prompt:**

*"For a paper about the impact of game-based learning on students' math performance, suggest the complete paper structure."*

The output might look like this:

**Title**

The Impact of Digital Educational Games on Elementary Students' Math Learning

**Abstract**

A summary of the research goal, method, main findings, and conclusion

**Keywords**

educational game, math learning, educational technology, elementary education

**Introduction**

- Stating the problem
- The importance of the topic
- A review of prior research
- The research goal and question

**Method**

- Research design
- Study population and sample
- Data collection tools

- Data analysis method

**Findings**

- Presenting descriptive statistics
- Statistical test results
- Tables and figures

**Discussion and conclusion**

- Interpreting the findings
- Comparing with prior research
- The research's limitations
- Suggestions for future research

**References**

Having a structure like this makes writing the paper much simpler and more organized.

## 2. Writing the Paper's Different Sections

Every section of a scientific paper has a specific purpose and needs to be written in a specific way. Claude can help the researcher write each section.

**Abstract**

The abstract is a summary of the whole paper and is usually between 150 and 250 words. Readers decide whether to read the full paper based on the abstract.

A good abstract usually includes four elements:

- The research goal
- The method used
- The main findings
- The conclusion

**Sample prompt:**

*"Based on this information, write a scientific abstract:*

*Goal: examining the effect of game-based instruction on math learning*
*Method: an experiment with two groups of 30*

*Statistical result: $t(58) = 3.12$, $p < 0.01$*
*Conclusion: the group that used the game had better performance."*

Claude can turn this information into a coherent abstract.

**Introduction**

The introduction's purpose is to introduce the research problem and demonstrate its importance. The introduction should lead the reader toward the research question.

A good introduction usually includes these steps:

- Introducing the topic
- Stating the topic's importance
- A brief review of prior research
- Stating the research gap
- Introducing the goal or research question

**Sample prompt:**

*"Write an opening paragraph for the introduction of a paper about using games in math education."*

The output might be:

*"Learning math is one of the important challenges in elementary education. Many students form a weak connection with mathematical concepts in the early years of schooling, and this can affect their academic progress in later years. In recent years, using digital educational games has drawn attention as a way to increase student motivation and engagement."*

**Method**

In this section, you need to explain how the research was conducted. This section's purpose is transparency and replicability.

There are usually four main parts to a method section:

- Research design
- Population and sample
- Data collection tools
- Data analysis method

**Sample prompt:**

*"Based on this information, write the method section:*

*Design: quasi-experimental*
*Sample: 60 third-grade students*
*Tool: a math achievement test*
*Analysis: an independent t-test"*

Claude can turn this information into formal scientific text.

**Results**

In this section, the results of the data analysis are presented. This part usually uses tables, figures, and statistics.

This section's purpose is only to report the results, not to interpret them.

**Sample statistical reporting:**

*"The results of the independent t-test showed that the mean score of the experimental group (M = 17.8) was significantly higher than the control group (M = 14.6), t(58) = 3.12, p < 0.01."*

Claude can turn your raw data or statistical results into standardized sentences like this.

**Discussion and conclusion**

In this section, the researcher interprets the results and explains what these findings mean.

This section usually includes:

- Interpreting the findings
- Comparing with prior research
- Stating the research's limitations
- Offering suggestions

**Sample prompt:**

*"Based on the finding that using educational games increased scores, write a discussion paragraph."*

Claude can explain why this method was effective and how it relates to prior research.

## 3. Editing and Preparing for Submission

After writing the paper, a very important stage begins, called scientific editing. Even good papers can look weak without careful editing.

Claude can help with several types of editing.

**Language editing**

- Fixing complex sentences
- Improving the scientific tone
- Shortening the text

**Sample prompt:**

*"Rewrite this paragraph in more scientific, smoother language."*

You can also ask Claude to find problems in the paper.

**Sample prompt:**

*"Review this section of the paper and tell me what problems it has."*

The answer might include things like:

- Overly long sentences
- Repetition of certain concepts
- Lack of clear connection between paragraphs

**Important points in final editing:**

- Checking the consistency of tone across all sections
- Making sure technical terms are used correctly
- Checking the accuracy of citations and references
- Controlling the formatting of tables and figures
- Checking compliance with the target journal's guidelines

## Summary

A scientific paper is the most important tool for publishing research results. In this chapter, we saw that Claude can help at different stages of writing a paper:

- Designing the paper's structure
- Writing the different sections
- Turning data into a scientific report

- Editing and improving the text

Using this tool properly, researchers can turn their research into a paper faster, more precisely, and more professionally.

**Key takeaways from this chapter:**

- The IMRaD structure is the basic foundation of scientific papers
- Each section of the paper has its own purpose and writing style
- The abstract should be a complete summary of the whole research
- Scientific editing matters as much as the initial writing

---

# Chapter 33: Preparing for Your Defense

After writing the thesis or proposal, a final and critical stage arrives: the defense. The defense is a session in which the researcher must present their research to their advisor, co-advisor, and examiners, and answer their questions.

Many students feel stressed about the defense. Worrying about tough questions, fear of forgetting material, or not having a good answer — all of this is normal.

But with proper preparation, the defense can become a successful experience. Claude can give you a lot of help at this stage: from designing slides to practicing answers to likely questions.

In this chapter, we'll cover three main topics:

- Building defense slides
- Practicing answers to questions
- Mental and practical preparation

## 1. Building Defense Slides

Slides are your main presentation tool. Good slides should be simple, clear, and engaging. Bad slides can make even excellent research look weak.

**General principles of slide design**

- One main message per slide
- Less text, more visuals
- A readable, large font

- Simple, professional colors
- Clear tables and charts

Claude can help you design the slide structure.

**Sample prompt:**

*"I want to build slides for defending my thesis about the impact of online education on student learning. Suggest the overall slide structure."*

The output might look like this:

**Slide 1: Title**

- The thesis title
- The student's name
- The advisor
- The defense date

**Slide 2: Introduction**

- Stating the problem
- The topic's importance

**Slide 3: Research question**

- The main research question
- The hypotheses

**Slide 4: Literature review**

- A summary of prior research
- The research gap

**Slide 5: Research method**

- Research design
- Population and sample
- Data collection tools

**Slide 6: Findings (1)**

- The main statistical results
- A table or chart

**Slide 7: Findings (2)**

- Additional findings

**Slide 8: Discussion**

- Interpreting the results
- Comparison with prior research

**Slide 9: Conclusion**

- A summary of the findings
- Practical implications

**Slide 10: Limitations and suggestions**

- The research's limitations
- Suggestions for future research

**Slide 11: Thank you**

- Thanking the professors and participants

This structure can be the initial framework for building the slides.

**Content for each slide**

Once the structure is designed, you need to prepare the content of each slide. Claude can help you write each slide's text so it's concise and clear.

**Sample prompt:**

*"For the introduction slide, write a short, engaging text that introduces the research problem. Topic: the impact of online education on student learning."*

The output might be:

**The research problem**

*"Online education has grown rapidly in recent years. But is this method as effective as in-person education? Do students learn well in an online environment?"*

This text is simple, clear, and thought-provoking.

**Designing the findings slide**

The findings slides are usually the most important part of the presentation. In this part, you need to show the research results clearly.

**Sample prompt:**

*"For the findings slide, design a simple table showing the t-test results.*

*Experimental group: mean 17.8*
*Control group: mean 14.6*
*t(58) = 3.12, p < 0.01"*

Claude can suggest a simple, readable table:

| Group | Mean | Standard Deviation |
|---|---|---|
| Experimental | 17.8 | 2.3 |
| Control | 14.6 | 2.1 |

Statistical result: t(58) = 3.12, p < 0.01

You can also ask Claude to write a short explanation for this table.

**Sample prompt:**

*"Write one explanatory sentence for this table."*

**Output:**

*"The experimental group scored significantly higher than the control group."*

## 2. Practicing Answers to Questions

One of the most important parts of preparing for a defense is practicing answers to likely questions. Professors might ask various things: about the research method, the limitations, interpreting the results, or even theoretical aspects of the topic.

Claude can help you anticipate likely questions and prepare suitable answers.

**Identifying likely questions**

**Sample prompt:**

*"My thesis is about the impact of online education on student learning. The research method was quasi-experimental with a sample of 60. What questions might I be asked at the defense?"*

The output might include these questions:

**Questions about the method:**

- Why did you choose a quasi-experimental method?
- How did you divide the groups?
- Did you control for confounding variables?

**Questions about the sample:**

- Why was the sample size 60?
- Is this sample representative of the population?
- Did you have any sample attrition?

**Questions about the findings:**

- Why did the experimental group perform better?
- Is this difference practically significant, or only statistically?
- Can these results be generalized to other populations?

**Questions about limitations:**

- What were the main limitations of your research?
- If you did this research again, what would you change?

**Practicing answers**

After identifying the questions, you can practice answering them with Claude.

**Sample prompt:**

*"Act as a thesis examiner and ask me questions. My research topic is the impact of online education on student learning."*

Claude can create a simple simulation of a defense session:

*Examiner: "Why did you choose a quasi-experimental method for this research?"*

*Student (a suggested answer): "Full randomization wasn't feasible due to practical constraints. So a quasi-experimental design with a control group was used to examine the impact of the educational intervention with reasonable precision."*

This kind of practice gives you more confidence during the actual defense.

## 3. Mental and Practical Preparation

Beyond the slides and answering questions, mental readiness matters a great deal too.

A few important tips for a successful defense:

- Practice your presentation several times
- Control your presentation time
- Don't memorize the slides — understand their concepts
- Have a simple explanation ready for every chart or table
- Get familiar with the defense room's setup
- Wear suitable, formal clothing
- Get enough rest the night before

Claude can even help you practice your spoken presentation text.

**Sample prompt:**

*"Based on these slides, write a short text for a 10-minute presentation."*

Or:

*"Shorten this presentation text so it fits into a 7-minute talk."*

## Summary

The defense session is the final stage of presenting a research project. Success at this stage doesn't just depend on the quality of the research — it also depends on how well you present it and how prepared you are to answer questions.

In this chapter, we saw that Claude can help at several stages:

- Designing the defense slide structure
- Summarizing content for the presentation
- Anticipating examiners' questions
- Practicing answers to questions
- Preparing the presentation text

With proper practice and preparation, the defense session can become an opportunity to showcase your research abilities.

---

# Part Seven: Skills and Automation

# Chapter 34: The Concept of Skills

Up to this point, you've learned how to write prompts for Claude and use it for various tasks. But there's one problem: if you want to repeat the same task every day, you have to write the same prompt again, or copy it from somewhere.

To solve this problem, Claude has a capability called Skills, which makes your work much simpler.

## What Is a Skill?

A Skill is a reusable prompt: you write it once, save it, and then call it up with a single click whenever you need it.

Instead of writing this every time:

*"Summarize this text. The summary should be a maximum of 100 words and should include the key points."*

You build a Skill called "Professional Summarizer," and next time, you just click on it.

## Why Do Skills Matter?

**1. A big time saver**

If you need to summarize a piece of text ten times in one day, you don't need to write the prompt ten times. You build the Skill once, then just click.

**2. Consistency in output**

When you write the prompt manually each time, it might shift slightly, and you get a different output. But with Skills, the exact same instruction always runs, and the output is more consistent.

**3. Fewer errors**

You no longer have to worry about forgetting a part of the prompt or writing it wrong.

**4. Increased speed**

For repetitive daily tasks, Skills multiply your speed.

## The Structure of a Skill

Every Skill has two parts:

**1. The Skill's name**

The name you give it so you can easily find it later.

Example:

- Professional Summarizer
- Persian Text Editor
- Turning Text into Key Points

**2. The instruction (prompt)**

The prompt you want Claude to run every time.

Example:

*"Summarize the submitted text.*
*- The summary should be a maximum of 100 words*
*- Preserve the key points*
*- Use simple language"*

## How Do Skills Work?

1. You build a Skill and save it
2. When you want to use it, you click on the Skill's name
3. Claude runs the saved instruction
4. You just send the text or information needed

## A Practical Example

Suppose you need to read several English articles every day and write Persian summaries of them.

**Without Skills:**

Every time, you have to write:

*"Read this English article and write its Persian summary in 150 words. The summary should include the goal, the method, and the result."*

**With Skills:**

You build a Skill called "Summarize English Article to Persian" once, and every time after that, you just click on it and send the article.

## Applications of Skills in Research and Academia

- Summarizing scholarly papers with a fixed structure
- Editing Persian text against specific criteria
- Turning raw findings into a standard report
- Designing research questions based on a specific pattern
- Coding qualitative data with a fixed method

## An Important Note

Skills are suitable for tasks that are repetitive and have a fixed structure. If you want to do a different task each time, it's better to write a regular prompt.

In the next chapter, you'll learn how to build your first Skill and use it.

---

# Chapter 35: Building Personal Skills

In the previous chapter, we got to know the concept of Skills and saw how they can be used to save and run repetitive prompts. Now it's time to learn how to build a personal Skill.

If you're already familiar with writing prompts, building a Skill won't be hard for you. In fact, a Skill is really just your prompt, saved in an organized form so you can use it repeatedly without writing the instruction from scratch every time.

In this chapter, we'll cover three main topics:

- The steps to building a Skill
- Important points for writing a Skill's instructions
- A few examples of useful Skills

## Steps to Building a Skill

Building a Skill is usually done in four simple steps.

**Step one: Identify a repetitive task**

The first step is to find a task you do regularly, and for which you almost always write a similar instruction.

To find tasks like this, you can ask yourself:

- Do I do this task often?

- Do I write roughly the same kind of instruction each time?
- Does the output I want have a specific structure?

If the answer to these questions is yes, that task is a good candidate to turn into a Skill.

**Examples of tasks suitable for a Skill:**

- Writing a work email
- Summarizing an article
- Generating a social media post
- Translating text
- Writing a report

For example, suppose every week you need to write a project progress report. This report always has three sections: completed tasks, problems, and next week's plan. Since this is a repetitive task, a Skill can be built for it.

**Step two: Design and test the prompt**

Before building the Skill, it's best to use your intended prompt in the regular way a few times first and check the result.

**Sample prompt:**

*"Write an Instagram post about the given topic. The tone should be warm and engaging. Use a few suitable emojis. The text length should be about 100 to 150 words, and end with a question to engage the audience."*

If you've used this prompt several times and the outputs were good, you can turn it into a Skill.

**Step three: Define the Skill**

At this stage, your prompt gets saved as a Skill. Every Skill usually has two main parts:

**1. The Skill's name**

The name should be short, clear, and understandable.

Examples:

- Instagram Post
- Article Summary
- Formal Email

- English-to-Persian Translation

**2. The Skill's instructions**

In this part, you explain exactly what Claude needs to do.

**Sample instructions:**

*"Write an Instagram post about the given topic. The tone should be warm and engaging. Use a suitable emoji. The text length should be between 100 and 150 words, and pose a question at the end to increase audience engagement."*

Once the Skill is saved, you no longer need to write this instruction from scratch every time.

**Step four: Test the Skill**

After building the Skill, it's best to try it out. Give it a sample input and see whether the output matches your expectations.

**Sample input:**

*"Topic: the benefits of daily reading"*

If the result isn't suitable, you can edit the Skill's instructions to make the output more precise. Sometimes just adding a small bit of explanation makes the output quality much better.

## Important Points for Writing a Skill's Instructions

The quality of a Skill depends a great deal on the quality of its instructions. The clearer your instruction, the better the output will be.

**1. Write clearly**

Don't use vague instructions.

Weak example:

*"Write a text."*

Better example:

*"Write a 120-word Instagram post with a warm tone."*

**2. Specify the output structure**

If you want the output to follow a specific format, write that in the instructions.

Example:

*"The report should have three sections:*
*1. Completed tasks*
*2. Problems*
*3. Next week's plan"*

This makes the output always organized and predictable.

**3. Specify the tone**

If you have a specific style in mind, mention it.

Examples:

- Formal tone
- Warm tone
- Instructional tone
- Professional tone

Specifying the tone brings the result closer to what you have in mind.

**4. Set constraints**

Sometimes you need to specify the text length or specific conditions.

Examples:

- The text should be a maximum of 150 words
- Use simple language
- The output should be in the form of a bullet-point list

These constraints help the model produce a more precise output.

## A Few Examples of Useful Skills

**Skill: Article Summarizer**

Instructions:

*"Summarize the given text and extract only the most important points. Present the output as a few short bullet points."*

**Skill: Formal Email**

Instructions:

*"Write a formal, respectful email. The structure should include a greeting, the main text, and a closing."*

**Skill: Translation**

Instructions:

*"Translate the given English text into Persian. The translation should be smooth and natural, and technical terms should be preserved."*

## Summary

A Skill is really just a saved prompt that you can use repeatedly. By building personal Skills, repetitive tasks get done faster, and the quality of the output becomes more consistent too.

Whenever you notice you're writing the same instruction over and over, that's probably a sign it's time to turn it into a Skill. This saves time and makes working with AI models much simpler and more professional.

In the next chapter, we'll learn how to combine several Skills together and build more powerful workflows.

---

# Chapter 36: Combining Skills and Building Workflows

In the previous chapter, we learned how to build a personal Skill. But the real power of Skills becomes apparent when you combine several Skills together and build a complete Workflow.

In this chapter, we'll learn how to connect Skills to each other and automate more complex tasks.

Three main topics in this chapter:

- The concept of a Workflow and how it differs from a Skill
- Designing a Workflow
- Practical Workflow examples

## The Concept of a Workflow

A Workflow means a working flow: a sequence of steps performed in order to reach a final result.

**The difference between a Skill and a Workflow:**

**Skill:** performs one specific task.

Example: summarizing a text

**Workflow:** performs several tasks in sequence.

Example: summarizing a text, then translating it, then turning it into a social media post

**A simple example:**

Suppose you want to read an English article, summarize it, and turn it into a Persian Instagram post.

**Without a Workflow:**

1. You give the article to Claude and say "summarize it"
2. You copy the summary
3. You tell Claude "translate it"
4. You copy the translation
5. You tell Claude "make an Instagram post"

**With a Workflow:**

You just give it the article, and the Workflow automatically does all the steps.

## Designing a Workflow

Building a Workflow happens in four stages.

**Stage 1: Identify the work stages**

First, you need to figure out what stages your task consists of.

**Example:**

Task: generating social media content from an article

Stages:

1. Summarize the article
2. Extract key points

3. Write a post with the appropriate tone
4. Add hashtags

**Stage 2: Determine the order of the stages**

Specify which stage needs to happen first, and note that each stage's output is the input to the next stage.

**Example:**

Article → Summary → Key points → Post → Hashtags

This order is logical because you can't extract key points before summarizing.

**Stage 3: Build the required Skills**

Build one Skill for each stage.

**Skill 1: Summarizer**

*"Summarize the given text in 3 paragraphs. Keep only the most important information."*

**Skill 2: Extract Points**

*"Extract 5 key points from the given text. Each point should be one short sentence."*

**Skill 3: Post Writer**

*"Write a 120-word Instagram post from the given points. The tone should be warm and engaging."*

**Skill 4: Hashtag Generator**

*"Suggest 5 relevant, popular hashtags for the given post."*

**Stage 4: Connect the Skills**

Now you need to connect the Skills together to build a complete Workflow.

**How to connect them:**

On most platforms, you can specify that the output of one Skill be used as the input for the next Skill.

**Sample Workflow:**

Input: the original article
↓
Skill 1: Summarizer
↓
Skill 2: Extract Points
↓
Skill 3: Post Writer
↓
Skill 4: Hashtag Generator
↓
Output: a complete post with hashtags

## Important Points When Designing a Workflow

**1. Each stage should do one specific task**

Don't make your Skills too complex. It's better for each Skill to do one simple task.

Wrong:

A single Skill that summarizes, translates, and builds a post all at the same time.

Right:

Three separate Skills for summarizing, translating, and building the post.

**2. It should have a logical order**

Make sure the output of each stage is suitable for the next stage.

Wrong example:

Translate → Summarize

(If the original text is in English, it should be summarized first, then translated)

Correct example:

Summarize → Translate

**3. It should be editable**

Sometimes you might want to step in between two stages and change something. A Workflow should let you edit a stage's output before it's sent to the next stage, if needed.

**4. It should be reusable**

A good Workflow should be usable for similar cases.

Example: if you've built a Workflow for turning an article into an Instagram post, you should be able to use it for any article, not just one specific one.

## Practical Workflow Examples

**Workflow 1: Multilingual content generation**

Goal: build a post in three languages — Persian, English, and Arabic

Stages:

Original idea
↓
Skill 1: Write the Persian post
↓
Skill 2: Translate to English
↓
Skill 3: Translate to Arabic
↓
Output: three posts in three languages

**Workflow 2: Analysis and report writing**

Goal: turn raw data into a final report

Stages:

Raw data
↓
Skill 1: Analyze the data
↓
Skill 2: Extract key results
↓
Skill 3: Write a formal report
↓
Skill 4: Build an executive summary
↓
Output: the complete report

**Workflow 3: Preparing an article for publication**

Goal: edit and optimize an article

Stages:

Original article
↓
Skill 1: Language editing
↓
Skill 2: Simplify the text
↓
Skill 3: Optimize the title and subheadings
↓
Skill 4: Write the meta description
↓
Output: the article ready for publication

## Semi-Automatic vs. Fully Automatic Workflows

There are two types of Workflows:

**1. Semi-automatic**

Between stages, you review the output and revise it if needed. This method is more suitable for sensitive tasks like a scientific paper or a formal report.

**2. Fully automatic**

All stages run without your involvement. This method is suitable for repetitive, low-risk tasks like generating a daily post.

## Common Mistakes in Building a Workflow

**1. Building an overly complex Workflow**

If there are too many stages, quality control becomes hard.

**2. Combining several tasks into one Skill**

This reduces flexibility.

**3. Not having a standard output**

If the output of each stage doesn't have a clear structure, the next stage will run into errors.

## A Complete Example: From Idea to a Content Campaign

Goal: building a simple campaign to introduce a book

Input: a short description of the book
↓
Skill 1: Write a formal introduction of the book
↓
Skill 2: Extract 5 inspiring sentences
↓
Skill 3: Turn each sentence into a social media post
↓
Skill 4: Generate 10 relevant hashtags
↓
Output: a complete campaign content package

With this Workflow, you can produce several pieces of content from a single simple description.

## Summary

A Skill is a tool for doing one specific task. A Workflow is a combination of several Skills that runs a complete path.

By building a Workflow, you can:

- Reduce the time it takes to do tasks
- Keep the quality of output consistent
- Break complex tasks down into simple stages
- Make repetitive processes semi-automatic or fully automatic

Once you learn to put Skills together like Lego pieces, you can build powerful working systems that handle a large share of your daily tasks.

In the next chapter, we'll move on to more advanced automation and learn how to connect Claude to other tools to build a unified working system.

---

# Chapter 37: Slash Commands

Alongside Skills and Workflows, another tool that makes working with Claude faster and simpler is Slash Commands, or quick commands. These commands let you run a specific

action by typing a short phrase that starts with a `/`. Instead of writing a complete prompt every time, you can just type a short command and get the same result.

In this chapter, we'll first cover the concept of Slash Commands, and then look at a few practical examples of them.

## Part One: The Concept of Quick Commands

A Slash Command is really a shortcut for a complete instruction. When you make the same kind of request to Claude regularly, writing the same prompt over and over can be time-consuming. Slash Commands solve this problem. You define a short command once, and every time you type it, Claude knows what to do.

Suppose you summarize long texts every day. If you have to write "summarize this text in three paragraphs and keep only the important points" every single time, your work slows down. But if you have a Slash Command like `/summary`, all you need to do is type it and enter the text.

## The Benefits of Slash Commands

#### High speed

Instead of several sentences, you write just one word.

#### Consistency in output

Since the instruction is always the same, the result will also have a similar structure.

#### Fewer errors

When we write an instruction manually each time, we might change words or forget part of the instruction. But with a Slash Command, the instruction is always exactly what we defined.

#### Better organization

When you have several different Slash Commands, working with Claude becomes very much like working with a professional tool. Instead of writing long instructions, you just run a collection of shortcuts.

## The Difference Between Slash Commands and Skills

The important point is that Slash Commands are usually used for simple, repetitive tasks. If a task is complex and involves several stages, it's better to use Skills or a Workflow. But if the

goal is just quickly running a specific instruction, Slash Commands are a very good option.

| Feature | Slash Command | Skill |
|---|---|---|
| Access speed | Very fast (just type it) | Requires selecting from a menu |
| Complexity | Simple and short | Can be complex |
| Use case | Repetitive daily tasks | Structured tasks |
| Combinability | Usually standalone | Can be placed inside a Workflow |

## Principles for Choosing a Good Slash Command

There are a few simple principles for choosing a good Slash Command:

**1. Its name should be short**

Example: `/sum` instead of `/please_summarize_this_text`

**2. Its meaning should be clear**

Example: `/translate` for translating, `/fix` for correcting

**3. It should be easy to remember**

Example: `/email` for writing an email

**Good examples:**

- `/sum` for summarizing
- `/en` for translating into English
- `/fix` for editing

**Bad examples:**

- `/x` (unclear)
- `/task1` (unprofessional)
- `/please_do_this_for_me` (too long)

## Part Two: Practical Examples

**1. Summarizing text**

One of the most common uses of Slash Commands is summarizing text. Suppose you have a long article and want to quickly know its main points.

Command: `/summarize` or `/sum`

**How to use it:**

```
/sum
[put the article's text here]
```

Output: a short summary of the text's main points

**2. Quick translation**

Many users regularly translate text between different languages.

Command: `/translate-en` or `/en`

**How to use it:**

```
/en
Translate this text.
```

Output: *"Translate this text."*

In the same way, you can build other commands for translating into different languages:

- `/fa` for translating into Persian
- `/ar` for translating into Arabic
- `/fr` for translating into French

**3. Correcting and editing text**

If you have text that might have spelling or grammar mistakes, the `/fix` command can correct it.

Command: `/fix`

**How to use it:**

```
/fix
this text has typos and needs to be fixed
```

Output: *"This text has typos and needs to be fixed."*

Claude reviews the text and gives you a smoother, more correct version.

**4. Converting to a specific structure**

If you have scattered notes, you can use the `/list` command to ask Claude to turn them into an organized list.

Command: `/list`

**How to use it:**

```
/list
I need to go to the store. I need to buy bread. I need to buy milk. I
need to buy eggs.
```

Output:

1. Bread
2. Milk
3. Eggs

This is very useful for organizing information.

**5. Content generation**

Slash Commands are also useful for content generation.

Command for a social media post: `/post`

**How to use it:**

```
/post
Topic: the importance of daily reading
```

Output: text suitable for social media, with an engaging tone

Command for a formal email: `/email`

**How to use it:**

```
/email
Subject: requesting time off
Recipient: manager
Content: I want to take 3 days off
```

Output:

*"Dear Sir/Madam,*

*I hope you are doing well. I intend to take 3 days of my entitled leave starting on [date].*

*I would appreciate your approval.*

*Regards,*
*[Your name]"*

**6. Research tasks**

You can use these commands even in research work.

Command for extracting keywords: `/keywords`

**How to use it:**

```
/keywords
[the scientific paper's text]
```

Output: a list of important keywords

Command for a research summary: `/research-summary`

**How to use it:**

```
/research-summary
[the paper's text]
```

Output: the paper's main points, structured (goal, method, results, conclusion)

## Summary

Slash Commands are like quick buttons for AI. When you define a set of these commands for your daily tasks, working with Claude becomes much faster. Instead of writing long instructions, you just have a few short shortcuts, each of which performs a specific task.

The more intelligently you design these buttons, the more tasks you'll be able to do with greater speed and precision.

In practice, combining Slash Commands for quick tasks, Skills for structured tasks, and Workflows for multi-stage processes builds a powerful, efficient working system.

In the next chapter, we'll move on to more advanced topics and learn how to integrate Claude with other tools to build a unified working environment.

---

# Part Eight: Advanced Techniques

# Chapter 38: Smart Token Management and Workspace Optimization

One hidden problem that many Claude users run into is token waste. You might think only the text of your prompt consumes tokens, but the reality is that a significant portion of tokens gets used up before Claude has even started processing your request.

Boris Cherny, one of the lead engineers on the Claude Code project at Anthropic, revealed this in a recent podcast. He showed that in many cases, up to 73% of the tokens users consume goes to things that have no connection at all to their actual request.

This means that if you feel like Claude isn't working the way it used to, or you're hitting your usage cap too quickly, the problem is probably poor token management, not the quality of the model.

## The Main Sources of Token Waste

Based on the analyses that have been done, there are 9 mistaken patterns that cause token waste. The three main ones are:

**1. Project configuration files (like CLAUDE.md)**

If you're working on a project that has a custom configuration file, this file gets loaded every single time Claude runs. Even if its content isn't needed for your current request, it still consumes tokens. Statistics show that about 14% of tokens are wasted just because of these files.

**Solution:** if you have a project configuration file, keep its content brief. Only include the essential instructions in it, and avoid adding extra explanations or long examples.

**2. Old conversation history**

To better understand your request, Claude also reviews the history of previous conversations. But if the conversation has gotten very long, or includes different topics that no longer relate to the current task, this history just consumes extra tokens. About 13% of token waste is due to exactly this.

**Solution:** when the subject of your work changes, start a new conversation. Keeping one long conversation going for different topics not only consumes extra tokens, it can also make Claude focus less on your current request.

**3. Active hooks and plugins**

If you're using tools like Claude Code, you might have hooks or plugins installed that are active in the background. These tools consume tokens even if you don't currently need them. About 11% of token waste is due to this.

**Solution:** regularly check which hooks or plugins are active. If you don't need some of them, disable them.

## The Concept of Environment Cleanup

One professional technique advanced Claude users employ is cleaning up the working environment before starting a big task. This means that before you make an important request, make sure that:

- The conversation history is relevant to the current topic
- Extra files that aren't needed are closed
- Unnecessary hooks and plugins are disabled

This makes Claude focus all its processing power on solving your problem, rather than reading redundant or unnecessary information.

## Why Does This Matter?

When tokens aren't wasted, you get two main benefits.

First, you pay less, or you hit your usage cap later. If you run into a token limit several times a week, you probably have at least a few of these harmful patterns in your setup.

Second, the quality of Claude's answers improves. When the model doesn't have to spend its time processing unnecessary information, it can think more deeply about your problem and give a more precise answer.

Many users who think "Claude isn't like it used to be" are actually victims of poor token management. The model's quality hasn't changed, but how it's used has a direct impact on the result.

### Token Optimization Checklist

Before starting an important task with Claude, check these items:

- Is the current conversation history relevant to the topic? If not, start a new conversation.
- Are there extra files open that aren't needed? Close them.
- Is your project configuration file brief and useful? If not, edit it.
- Are unnecessary hooks or plugins active? Disable them.

By following these simple points, you can reduce token waste by up to 70% and have a better experience working with Claude.

---

## Chapter 39: Combining Claude with Other Tools

One of the most powerful ways to use Claude is combining it with other tools. Claude on its own can do a lot, but when you combine it with other services and software, you can build more complex working processes whose different parts run automatically.

In this chapter, we'll first look at the concept of combining several tools into one Workflow, and then use a practical example to show how this combination can be applied in real work.

### Combining Several Tools in One Workflow

When you use Claude on its own, you're usually doing one specific task: summarizing text, writing an email, or reviewing code. But in many real-world situations, your task isn't just one stage. You might want to first fetch data from a source, then analyze it, write up the result as a report, and finally send it to another system.

This is where the concept of a multi-stage Workflow comes in. You can combine Claude with other tools — like databases, cloud services, data analysis software, project management tools, and even other AI models — to build a complete system.

**A simple example:** suppose you have a customer support team. Every day, you receive dozens of emails from customers. You could build a Workflow that:

1. Fetches emails from the inbox
2. Has Claude read them and identify the main topic of each email
3. Sorts each email into the appropriate category (technical, billing, general) based on the topic
4. Prepares a draft response for each category
5. Sends the response to the relevant team for final review

In this Workflow, Claude only handles part of the work. Fetching the emails is done by an email service, categorizing and analysis is done by Claude, and sending it to the team is done by a task management tool like Trello or Asana.

## Why Is Combining Tools Useful?

One benefit of this approach is that you can use each tool's strengths. Claude is excellent at analyzing text and generating content, but for storing data or automatically sending emails, other tools are better suited. By combining these tools, you build a powerful system that can perform complex tasks automatically.

## Connection Tools

To build these Workflows, tools like Zapier, Make (formerly Integromat), n8n, or even Claude's direct API are usually used. These tools let you connect different services together without needing complex programming.

For example, in Zapier you can build a "Zap" that, when a new email arrives in Gmail, sends it to Claude, receives Claude's response, and then saves it in a Google Sheet. All of these steps happen without writing a single line of code.

If you know how to program, you can use the Claude API and build more customized Workflows. The API lets you send your requests directly to Claude and process the response within your own app or system. This method is more flexible, and you can tune it exactly to your needs.

## Important Points in Designing a Workflow

When you combine several tools, you need to pay attention to the flow of data. Each stage needs to produce a suitable output that the next stage can process. For example, if Claude

generates a piece of text, you need to make sure the format of that text is compatible with the next tool's required input.

You also need to think about error handling. If one of the Workflow's stages runs into a problem, what should happen? Does the whole process stop, or does the next stage continue with incomplete data? Designing a robust Workflow means anticipating these situations and defining the appropriate behavior for each case.

## Practical Example: Building an Automatic News Blog

Now, with a complete example, we'll show how Claude can be combined with other tools. Suppose you have a blog and want to publish a daily summary of news related to your field. Doing this manually is time-consuming, but you can automate it.

**Stage one: Gathering the news**

First, you need to gather news from various sources. You can use an RSS service like Feedly, or a news site's API. For example, suppose you use the Google News RSS feed. This service gives you a list of news headlines and their corresponding links.

At this stage, you build a simple script or a Zap in Zapier that checks the RSS feed every morning and extracts the top 10 news items. The output of this stage is a list of headlines and links.

**Stage two: Analysis and summarization with Claude**

Now you send this list to Claude. Your prompt could look something like this:

*"I have a list of 10 news items. Please:*
*- Remove duplicate or low-importance items*
*- Select the 5 important news items*
*- Write a 2-3-line summary for each item*
*- Present the output as a simple list"*

Claude does this and gives you a summarized list of the important news.

**Stage three: Generating the final content**

At this stage, you can ask Claude to write a complete piece of text for the blog. Your prompt could be:

*"Using these 5 news items, write a short blog post that includes:*
*- An engaging introduction*

*- A summary of each news item with its corresponding link*
*- A concluding sentence*

*The tone should be formal but friendly."*

Claude generates a complete text ready for publication.

**Stage four: Automatic publication**

Finally, you can automatically publish this text on your blog. If you use WordPress, you can use its API to create a new post. Or if you want to review the text before publishing, you can send it to your email or save it in a Google Doc.

## The Result

With this Workflow, a task that might previously have taken several hours now gets done in a few minutes. You only need to set up the Workflow once, and after that it runs automatically every day.

This example shows how Claude can act as one part of a larger system. You can apply this idea to other tasks as well: analyzing sales data, generating weekly reports, automatically answering FAQs, and many other things.

Final note: combining Claude with other tools is one of the most important professional skills. By learning this method, you can automate many repetitive tasks and spend your time on more creative and important work.

---

# Chapter 40: Data Security

When you use Claude or any other AI tool, paying attention to data security is very important. In many cases, users hand information over to the system that might include personal, organizational, or business data. If this information isn't managed properly, it can create security problems or privacy violations.

In this chapter, we'll cover two main topics: managing sensitive information, and best security practices when using Claude.

## Managing Sensitive Information

The first security principle is to always be careful about the information you send. It's best not to enter confidential or highly sensitive information directly into AI systems.

**What information is sensitive?**

Examples of sensitive information include:

- **Personal information:** national ID numbers, bank card numbers, people's exact addresses, personal phone numbers
- **Confidential organizational information:** business contracts, strategic plans, internal financial information
- **Security information:** passwords, API keys, access tokens
- **Medical or financial data:** patient records, customers' bank account information

If you need to work with this kind of data, it's best to anonymize it first. For example, you can remove people's names or replace sensitive data with generic identifiers.

**Practical example: Anonymizing data**

Instead of sending this text:

*"The company's financial record includes bank account number 1234-5678-9012 and a contract with client Mr. Ahmadi, national ID 0012345678."*

You could write:

*"A manufacturing company has several client contracts, and I want you to analyze the structure of its financial report. The information includes account numbers and client IDs."*

In this case, you've explained the problem without disclosing the real, sensitive information. Claude can still help you analyze the structure, but no confidential data has been handed to the system.

## Using Sample Data

One of the best ways to work with sensitive data is to use sample or fabricated data. For example, if you want to build a financial data analysis formula, you can first test it with hypothetical numbers, and once you're confident the method is correct, run it on the real data.

**Example:**

*"I want to build a formula for calculating net profit.*

*Sample data:*
*- Total revenue: 100 million tomans*

*- Expenses: 60 million tomans*
*- Tax: 10%*

*Please write the formula and test it on this data."*

This method lets you design and test your workflow without exposing real information.

## Best Security Practices

Beyond removing sensitive information, a few simple practices can increase the security of working with Claude.

**1. Review file content before sending**

Before sending files, check whether the entire content of the file is needed for the analysis. If only part of the file is needed, it's better to send just that part.

Example: if you want to edit one chapter of a thesis, send just that chapter, not the entire thesis file, which might include personal information or confidential research data.

**2. Use test versions**

Instead of using real data, you can create sample or fabricated data and test the analysis process on it first. This method is especially useful for complex or sensitive tasks.

**3. Control access within organizations**

If you're using Claude within an organization, it's best to specify who has access to what kind of data. This prevents information from being unintentionally spread.

For example, you could:

- Limit access to financial data to only the finance team
- Make customer information available only to the support team
- Avoid sharing conversations that contain sensitive information with other team members

**4. Read the privacy policies**

Always read the privacy policy of the service you're using. These policies explain how data is stored or processed and what restrictions exist on its use.

For example, Anthropic (the maker of Claude) has stated that:

- User data isn't used to train the model (except in specific cases and with the user's consent)

- Conversations are stored encrypted
- Users can delete their conversation history

That said, it's best to always assume that anything you send might be stored, and make decisions based on that assumption.

**5. Regularly delete conversation history**

If you've entered sensitive information in your conversations, it's best to delete those conversations once the work is done. This prevents sensitive information from accumulating in the history.

**6. Use separate environments**

For sensitive tasks, you can use separate user accounts or projects. This lets you keep personal and professional work separate, and if needed, delete just one of them.

## Security Checklist

Before sending information to Claude, check these items:

- [ ] Does this information include personal or confidential data?
- [ ] Can I anonymize this information?
- [ ] Can I use sample data instead of real data?
- [ ] Am I only sending the needed part of the file?
- [ ] Will I delete this conversation once the work is done?

Final note: AI tools are designed to help with analysis and content generation, but responsibility for managing data still rests with the user. By following a few simple security principles, you can enjoy the benefits of these tools without creating potential risks to information. Data security is a habit, not a one-time task.

---