

FREE SAMPLE

Claude Code for Non-Coders

*Build Websites, Tools and
Automations Without Writing Code*

*The full contents, the Four Habits, a first project
checked by hand, and what to say yes or no to*

Pranav S.

FREE SAMPLE · OCTOBER 2026

Contents of the full book

Everything in Claude Code for Non-Coders, section by section. The parts marked INCLUDED are complete in this sample, and those marked EXCERPT are partly included. Every chapter also ends with prompts to copy, common mistakes, a 10-minute exercise and key takeaways.

How to Use This Book

Who it's for · What you'll still do yourself · What you need · How to read it · Conventions · The example projects · A note on versions

PART I: START HERE

CHAPTER 1

What Claude Code Can Do for You EXCERPT

Claude Code in plain words · The Claude app or Claude Code? · What people in your role build · What you still own · Where it falls short · Who will rely on it? · The idea this book keeps coming back to · The Four Habits · How this book is organized

CHAPTER 2

Set Up in 20 Minutes, No Terminal

What you need · Install the app and sign in · Make a project folder · A tour of the app · Permission modes in plain English · Two setup traps · Pick a model · Check it works

CHAPTER 3

Your First Project: A Quote Calculator in 30 Minutes EXCERPT

Pick a good first project · Step 1: Describe it · Step 2: Let Claude plan · Step 3: Watch it build · Step 4: See it in the Browser pane · Step 5: Check it with known numbers · Step 6: Ask for a change · Step 7: Save a working version · Step 8 (optional): Put it on your phone · If it goes wrong · What just happened

PART II: WORKING WELL WITH CLAUDE

CHAPTER 4

Describing What You Want

Why vague requests cost you · The four parts of a good brief · Before and after: six requests · Show, don't just tell · Let Claude interview you · One request, one result · Steering mid-task

CHAPTER 5

Checking the Work Without Reading Code EXCERPT

Test what it does, not how it's written · Use it yourself · Known answers · Ask Claude to prove it · A second opinion: code review · Explanations at your level · Reading the Changes card without reading code · Seeing what Claude actually did · Red flags in Claude's replies · A one-page checking checklist · What to keep knowing yourself

CHAPTER 6

Project Notes and Saved Versions

Project notes: teach it once · Saved versions: going back when something breaks · Start fresh between jobs · Backing up (for anything that matters)

CHAPTER 7

When Things Go Wrong

Errors in plain English · When Claude goes in circles · "It worked yesterday" · If you never saved a version · It changed things you didn't ask for · Running into usage limits · When to stop and ask a developer

PART III: PLAYBOOKS

CHAPTER 8

Founders and PMs: Prototypes and MVPs EXCERPT

Prototype or product? · A clickable prototype for user interviews · Sharing it as an artifact · A landing page with a waitlist · Internal tools and demos · A spec Claude can build from · Handing off to a developer · Where not to do it yourself

CHAPTER 9

Operators and Marketers: Automate the Repetitive Work

Claude Code or the Claude app? · Spotting good candidates · Spreadsheet cleanup · The weekly report · Keeping it working when the input file changes · A campaign report for marketers · Drafting emails and content from data · Running it on a schedule · When a scheduled run fails · A live dashboard as an artifact

CHAPTER 10

Creators and Freelancers: Websites and Small Tools

A portfolio site from brief to Browser pane · Making it look right · Small tools for clients or your audience · Sharing a page or putting a site online · Edits after launch · Client work

CHAPTER 11

Analysts: Data Files, Charts and Repeatable Analysis

Ask for a script, not a one-off answer · Getting to know a dataset · Cleaning you can audit · Charts, summary tables and a shareable dashboard · Checking the numbers · Next month's file · Sensitive data

PART IV: GOING FURTHER, SAFELY

CHAPTER 12

Skills, Connectors, Schedules and the Cloud

Skills: a routine with a name · Connectors and plugins · Schedules · When you're ready

CHAPTER 13

Safety, Privacy and Cost **EXCERPT**

Permission requests: yes, no or "explain first" · A dedicated folder and backups · Passwords and keys · Personal and client data · Anything involving money · What goes public · Before anyone else relies on it · Spending less of your plan · Ten mistakes to avoid

AFTERWORD

Keep Going

The short version · Your first month · A little vocabulary worth learning · The four books · About this edition · A month from now

APPENDICES

APPENDIX A

Plain-English Glossary

A-C · D-F · G-L · M-P · R-S · T-W

APPENDIX B

45 Prompts to Copy **EXCERPT**

Starting a project · Describing a design · Checking work · Fixing problems · Data and spreadsheets · Websites and pages · Automations and schedules · Handing off to a developer · Safety and privacy

APPENDIX C

Project Notes Templates

How to use a template · Template 1: Prototype · Template 2: Automation · Template 3: Website · Template 4: Data analysis · Keeping the notes useful

APPENDIX D

Troubleshooting in Plain English

Start with these checks · Setup and sign-in · While Claude works · The result looks wrong · Schedules, skills and going live · Saving and going back · Limits and billing · Setting up a scheduled task with the form · Desktop app quick reference

APPENDIX E

Sources

Chapter 1: What Claude Code Can Do for You · Chapter 3: Your First Project: A Quote Calculator in 30 Minutes · Chapter 4: Describing What You Want · Chapter 5: Checking the Work Without Reading Code · Chapter 6: Project Notes and Saved Versions · Chapter 7: When Things Go Wrong · Chapter 8: Founders and PMs: Prototypes and MVPs · Chapter 9: Operators and Marketers: Automate the Repetitive Work · Chapter 10: Creators and Freelancers: Websites and Small Tools · Chapter 11: Analysts: Data Files, Charts and Repeatable Analysis · Chapter 12: Skills, Connectors, Schedules and the Cloud · Chapter 13: Safety, Privacy and Cost

What Claude Code Can Do for You

By the end of this chapter you'll know what Claude Code is, when it's the right tool and when it isn't, what stays your job, how careful to be before others rely on what you build, and the Four Habits the rest of this book builds on.

After a death in the family, a customer asked an airline's website chatbot about bereavement fares. It told the customer they could claim the discount after booking. The airline's policy didn't allow that, though the chatbot also linked to the correct policy page. Before a Canadian small-claims tribunal in 2024, the airline argued, in effect, that the chatbot was responsible for its own actions. The tribunal called this "a remarkable submission". It held that the airline is responsible for all the information on its website, and ordered it to pay about C\$812.

If your tool tells a customer something, you told them. That idea runs through "What you still own", later in this chapter.

Claude Code in plain words

Claude Code is an AI assistant that works on the files in a folder on your computer. You tell it in plain English what you want. It writes the code, runs it, checks what happened and fixes what went wrong, then tells you what it did or

asks you a question.

You never have to write code or understand it. You do have to do two things well: say clearly what you want, and check that what comes back actually works. Most of this book is about those two skills.

Say you run Maple Street Window Cleaning, a small business that quotes jobs at the customer's door. You open the Claude desktop app and type:

PROMPT

Make a one-page quote calculator for my window-cleaning business. Windows are \$8 each, screens \$3 each and gutter cleaning a flat \$120, plus 8% sales tax. When you're done, test it yourself: 12 windows, 12 screens and gutters should come to \$272.16.

Before reporting back, Claude might:

- Create a web page in your project folder.
- Try 12 windows, 12 screens and gutters on the page.
- Notice that the total comes to \$142.56 instead of \$272.16, because the gutter fee was left out, and fix it.
- Check again and see \$272.16.
- Tell you it's done, and that you can click the page's name to try it in the app's built-in browser (the Browser pane).

That loop of doing, checking and fixing, until the result matches your request or Claude needs you, is what makes Claude Code different from a plain chat that hands you text and leaves the trying-out to you.

NOTE: Claude Code also runs in a terminal (a text-only window where you type commands), which many programmers use. This book uses the **Code tab** of the Claude desktop app instead, so you never need a terminal.

The Claude app or Claude Code?

Anthropic makes two tools that sound alike, and the desktop app holds both:

- **The Claude app** is the everyday Claude you talk to, on the web, your desktop or your phone. It answers questions and writes drafts and plans. It can also take a one-off job off your hands and work across your files and tools: re-search, documents, spreadsheets, tidying folders. Jobs on files on your computer need the desktop app. In the merged app, in the words of Anthropic’s help article, “what you could previously only do in Claude Cowork is available from any conversation.” If your app still shows separate **Chat** and **Cowork**, use Cowork for these jobs.
- **Claude Code** lives in the desktop app’s **Code tab**, and it’s still separate. In Anthropic’s words, it’s “built for software engineering”. For you, that means using it when the result is something that *runs*: a web page, a calculator, or a small program that does the same job every week.

When this book says “the Claude app”, it means Claude’s everyday conversations, not the Code tab.

NOTE: Anthropic announced on September 16, 2026, that chat and Cowork are merging into one Claude. The change is rolling out gradually, starting with Pro and Max plans. On Team plans it’s in beta, and on Enterprise plans admins control when it appears. So your app may show one merged Claude, or separate **Chat** and **Cowork**. Either way, this book calls it “the Claude app”.

Anthropic doesn’t publish a rule for choosing between the Claude app and Claude Code. Here’s this book’s rule of thumb:

If you want...	Start with
An answer, an explanation or a first draft	The Claude app
A one-off job across your files, such as summarizing a folder of reports	The Claude app
Something that runs: a web page, a calculator, a dashboard	Claude Code

If you want...

Start with

The same job done exactly the same way every time

Claude Code

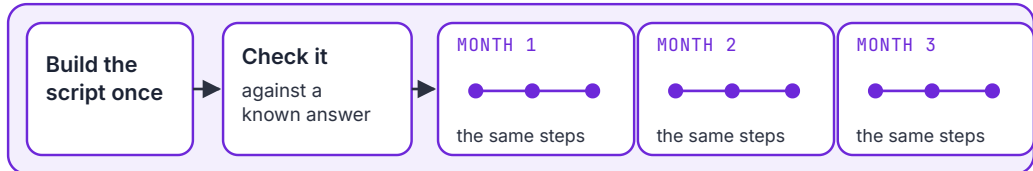
The last row needs explaining, because the Claude app can run scheduled tasks too. A scheduled task there does the job afresh, using Claude's judgment each time.

Claude Code instead builds you a tool, often a short program called a script, that follows the same steps every time. Once you've checked it, you know how next month's numbers will be worked out.

THE CLAUDE APP · A SCHEDULED TASK



CLAUDE CODE · A SCRIPT



A scheduled task in the Claude app does the job afresh each time, using Claude's judgment. A script built with Claude Code is checked against a known answer, then follows the same steps every month.

If you can't decide, ask: "Do I want a *thing* I'll use again?" If yes, use Claude Code. If not, start in the Claude app.

What you still own

However good the tool gets, three jobs stay yours: you decide what to build, you check that it's right, and you answer for it. In practice, that means these decisions, and this book keeps coming back to them:

- **What to build and why.** Claude builds what you describe, even when it's the wrong thing. Only you know what problem you're solving.
- **Whether it works.** Claude checks a lot on its own, but you know what "right" looks like for your business.
- **Whose data goes in.** Customer lists, client files, survey responses: you decide what Claude sees.
- **Anything involving money.** Taking payments, prices other people rely on, paid accounts and services.
- **What goes public.** Publishing a site or sharing a link is your decision, and you do it. Whatever your tool tells people is yours too.

Who will rely on it?

How careful you need to be depends on who will rely on what you build. This book sorts projects on a ladder of four levels, and each playbook in Part III names its level. Check again whenever more people start relying on a project.

Unsure between two levels?

Pick the higher one.

Check again whenever more people start relying on it.

Build it yourself.

1 · Just me

Only you use it, with your own or fake data

A quote calculator for your own jobs

Build it yourself, then go through “Before anyone else relies on it” (Chapter 13).

2 · My team

A few people you know

A weekly report your boss reads

Build it yourself if it only shows information. If it collects anything, add the privacy basics.

3 · The public

Strangers see or use it

A website or a sign-up page

4 · Money or sensitive data

Payments, logins, other people's personal data

A booking page that takes deposits, or a customer list you analyze

Prototype it with fake data. Launch only after a developer's review. Real personal data only after the checks in Chapters 11 and 13.

More care at each step

The “Who will rely on it?” ladder: four levels, and what each one asks of you before anyone relies on what you build.

Level	Example	What to do
1. Just me: only you use it, with your own data or fake data	A quote calculator for your own jobs	Build it yourself
2. My team: a few people you know rely on it	A weekly report your boss reads	Build it yourself, then go through “Before anyone else relies on it” (Chapter 13)
3. The public: strangers see or use it	A website or a sign-up page	Build it yourself if it only shows information. If it collects anything from visitors, also follow the privacy basics (Chapters 8, 10 and 13)
4. Money or sensitive data: payments, logins, or other people's personal data that you store or work with yourself	A booking page that takes deposits, or a customer list you analyze	Prototype it yourself with fake data. Launch only after a developer's review (Chapter 8), and let real personal data in only after the checks in Chapters 11 and 13

A waitlist kept by a mailing-list service stays at level 3, because the addresses never sit in your project (Chapter 8). And if the business can't run without a tool, decide who fixes it when it breaks, and have a developer review it, whatever its level.

Think of it as who you're cooking for. Cooking for yourself, you can improvise. For your family, you remember who won't eat onions. At a bake sale, strangers need labels saying what's inside. A restaurant that promises a dish is nut-free is making an allergen claim, and it gets inspected. Same kitchen, more care at each step.

If you're unsure between two levels, pick the higher one.

Can I check it? Can I undo it?

Beside “Who will rely on it?”, ask two more questions. The more people rely on a tool, the more a yes to both matters.

- **Can I check it?** Can a known answer, a second way of adding it up or a person tell a right result from a wrong one? If not, find a check before anyone relies on the tool (Chapter 5), or bring in a developer (Chapter 7).
- **Can I undo it?** Can you put a mistake right, or catch it before it reaches anyone? A changed file can go back to a saved version, but a sent email or a payment can't. Keep steps like those yours: Claude saves a draft, and you send or pay yourself (Chapters 9 and 13).

The idea this book keeps coming back to

Check it like a user, not a programmer.

You don't need to read code to know whether something works. You need to use it the way a real person would and compare what happens with what you expected. Programmers check the code. You check the result.

You'll do that in four ways throughout the book:

- **Use it yourself.** Click through it in the Browser pane, type into every box, press every button.
- **Try the awkward cases.** An empty box, a huge number, a file in the wrong format, a phone-sized screen.

- **Compare it with a known answer.** Work one quote out by hand, or check one total against your spreadsheet.
- **Ask Claude for proof.** “Run it and show me the result.” “Tell me exactly what you tried and what happened.”

One more trick does the most work: tell Claude how you’ll check the result *before* it starts, as the window-cleaning prompt does with its \$272.16. Then Claude can run the check itself and fix problems before you see them. Chapter 5 covers checking in depth.

The Four Habits

The rest of this book builds on four habits you can start today. They adapt Anthropic’s own best-practices guide for Claude Code for readers who don’t write code. Each gets a fuller treatment later.

Habit 1, Brief it

Give Claude a brief: say what you want, who it’s for, what must and mustn’t happen, and how you’ll know it works, ideally with one worked example.

A vague request makes Claude guess. It fills the gaps with sensible defaults that aren’t *your* defaults, and you spend the next hour correcting them. Compare:

PROMPT

Make me a quote calculator.

PROMPT

Make a one-page quote calculator for my window-cleaning business. Windows are \$8 each, screens \$3 each, gutters a flat \$120, plus 8% sales tax. I’ll use it on my phone at the customer’s door, so make the buttons and the total large. Check that 12 windows, 12 screens and gutters come to \$272.16.

The second takes a minute longer to write and saves several rounds of “no, not like that”. Chapter 4 is all about writing requests like it.

Habit 2, Teach it once

When you correct Claude, have it write the correction into the project notes so you never have to say it again.

A new session doesn’t see your earlier conversations. Claude may keep a few notes of its own, but you can’t count on them. Rules you rely on belong in your project notes: a file in your project called `CLAUDE.md`, which Claude reads at the start of every session.

Each session goes on cold, like an understudy who missed yesterday’s rehearsal. What carries the show is the prompt book, the stage manager’s master copy of every cue and change.

If you tell Claude “always show money with two decimal places” and leave it there, you may have to say it again tomorrow. Ask Claude to add it to the notes, and it’s a rule from then on. Chapter 6 shows you how.

Habit 3, Start fresh

When you move to something unrelated, start a new session.

A session carries its whole conversation with it, including each file Claude opened, each error and each false start. Once an old job’s leftovers crowd the new one, Claude makes more mistakes and uses more of your plan.

A new session clears the table. Your files and project notes are all still there. Chapter 6 covers this too.

Habit 4, Save points

When something works, make a save point: save a version before you change anything else.

You don't need a button for this: ask Claude to save a version. A saved version records exactly how your project looked at that moment. Programmers call it a *commit*.

If the next change breaks something, you can go back to the version that worked. Going back throws away anything you haven't saved, so Chapter 6 shows how to do it safely. Without saved versions, one bad change can undo an afternoon's progress.

TIP: If you remember one thing from this chapter, make it this: tell Claude how you'll check the result.

NOTE: The full chapter also covers what people in your role build, where Claude Code falls short and how the rest of the book is organized.

Your First Project: A Quote Calculator in 30 Minutes

In this chapter you'll build a working quote calculator from one written brief, check it against numbers you worked out yourself, change it and save a version you can always go back to. Along the way you'll learn the loop that carries you through every project in this book: describe, build, check, save.

Imagine quoting a job the old way. You're on a customer's doorstep with a clipboard, counting windows as the rain starts. Twelve windows, twelve screens, and the gutters look bad, so add those. You work the sum in the margin, add the tax and round somewhere along the way. The customer waits. Your pen smudges. Did you count the screens twice? You give a number and hope it's right. By the end of this chapter, that sum lives on your phone and updates as you tap, and you'll have checked it yourself, line by line, against numbers you trust.

Step 5: Check it with known numbers

Claude says the page gives \$272.16. Now check it yourself, working from paper or a calculator app, not from anything Claude told you:

Line	Working	Amount
Windows	12 × \$8	\$96.00

Line	Working	Amount
Screens	$12 \times \$3$	\$36.00
Gutters	flat fee	\$120.00
Subtotal	$96 + 36 + 120$	\$252.00
Tax	8% of \$252.00	\$20.16
Total	$252.00 + 20.16$	\$272.16

Enter 12 windows, 12 screens and gutters Yes in the page. Compare every line, the total included. The customer sees every line, so a wrong subtotal is a wrong quote even when the total looks right.

One check can pass by luck, so try a second job that's different in every way: 5 windows, no screens, no gutters. That should be a subtotal of \$40.00, tax of \$3.20 and a total of \$43.20. A page that always adds the gutters, or counts screens that aren't there, fails this one.

If a number doesn't match, tell Claude exactly what you entered, what you expected and what you saw:

PROMPT

The numbers don't match. I entered 5 windows, 0 screens and gutters No. I expected a subtotal of \$40.00, tax of \$3.20 and a total of \$43.20. The page shows a total of \$172.80. Find out why, fix it, and check both of my test jobs again.

Keep your known answers in a note, because you'll use them after every change. Chapter 6 shows how to put them in your project notes so Claude runs them too.

Step 6: Ask for a change

Say you give regular customers a percentage off. You want a discount box, with the discount taken off before tax.

Work out the known answer before you ask. Here's a 10% discount on the same job:

Line	Working	Amount
Subtotal	as before	\$252.00
Discount	10% of \$252.00	\$25.20
After discount	$252.00 - 25.20$	\$226.80
Tax	$8\% \text{ of } \$226.80 = 18.144$	\$18.14
Total	$226.80 + 18.14$	\$244.94

Look at the tax line. Until now every amount came out to whole cents, but 8% of \$226.80 is \$18.144. Money shows only two decimals, so the right answer is \$18.14.

This is exactly where calculators slip. One might round up to \$18.15 and show a total of \$244.95. Another might show tax of \$18.15 next to a total of \$244.94, so the lines don't add up. You don't need to know how rounding works in code, only that the right answers are \$18.14 and \$244.94.

This is a small change, so stay in Manual and send:

PROMPT

Add a discount box to the calculator, as a percentage: 10 means 10% off. Take the discount off the subtotal before tax, and show the discount and the subtotal after discount as their own lines. Leave the discount box empty to start, and treat empty as no discount.

How I'll check: 12 windows, 12 screens and gutters with a 10% discount should show a discount of \$25.20, a subtotal after discount of \$226.80, tax of \$18.14 and a total of \$244.94. With the discount box empty, the same job should still come to \$272.16. Try both before you tell me it's done.

Say Claude's first attempt slipped on the rounding. Its reply might look like this:

OUTPUT

Added a Discount (%) box, empty to start.

Tested 12 windows, 12 screens, gutters, 10% off. My first try gave tax of \$18.15 because I rounded up. Fixed. Now it gives:

Discount	\$25.20
After discount	\$226.80
Tax (8%)	\$18.14
Total	\$244.94

With the discount box empty, the total is \$272.16.

On screen, \$18.15 looks perfectly reasonable. Without \$18.14 in your brief, neither you nor Claude would have had anything to compare it with. That's the job a known answer does: it turns "looks right" into "is right".

A cent sounds too small to matter, until a rounding rule repeats. In January 1982 a Canadian stock exchange started a new index at 1,000. It was recalculated about 2,800 times a day, and each time the last digit was dropped instead of rounded. By November 1983 the index had sunk to around 520. The market hadn't fallen: corrected, the index opened at about 1,099. No single slip was big enough to notice. A known answer that tests the rounding, like \$18.14, is what catches it.

Now repeat Step 5 with the discount table above, comparing every line. Then clear the discount box, check that \$272.16 comes back, and try your \$43.20 job once more.

Why check every line? Say the page took the discount off after tax instead of before. The total would still be \$244.94, because 10% off \$272.16 rounds to the same amount, but the tax line would show \$20.16 instead of \$18.14. Only the line-by-line check catches it.

NOTE: These are Steps 5 and 6 of 8. Before them, you describe the calculator, let Claude plan it, watch it build and open it in the app. After them, you save a working version and put the calculator on your phone.

CHAPTER 5 · EXCERPT

Checking the Work Without Reading Code

A one-page checking checklist

Surgery has tested checklists at scale. In an eight-hospital study published in 2009, deaths fell from 1.5% to 0.8%, and complications from 11.0% to 7.0%, after a 19-item surgical checklist was introduced. Later, when a Canadian province's policy encouraged all its hospitals to adopt checklists, a study found no significant change in deaths or complications. The studies don't settle why the same kind of list brought different results. This book's reading is that it comes down to use: a checklist helps when people treat it as a real pause, not a box to tick. Use the one below that way.

Copy this into a note. You can also ask Claude to add it to your project notes, so it runs the parts it can every time (Chapter 6).

The Four Habits, as checks

- **Habit 1, Brief it:** your request names how you'll check the result.
- **Habit 2, Teach it once:** every correction you made is in `CLAUDE.md`.
- **Habit 3, Start fresh:** this session is only about this job.
- **Habit 4, Save points:** there's a saved version from before this change.

When Claude says “done”

- Did Claude run your check? Read the reply for red flags: “should work”, skipped checks, numbers with no source.
- If a Changes card appears, click its line counts to open the diff view. Are the files and sizes what you’d expect? Is anything deleted?
- Open the result in the Browser pane, click everything, then try odd inputs: empty, zero, huge, negative, words.
- If people will use it on a phone, check it at a phone’s width.
- Compare at least one result with a known answer.
- Ask what Claude tried and what happened, and for a plain-English explanation of what changed.

For bigger changes

- Run the automatic tests, and ask whether any tests changed.
- Run `/code-review` and get the findings in plain English.

When it passes

- Save a version, and add any new rule you discovered to the project notes.

CHAPTER 8 · EXCERPT

Founders and PMs: Prototypes and MVPs

Where not to do it yourself

Level 4 on the ladder, Money or sensitive data, is where you stop building alone. Claude will build any of these if you ask. That’s the problem. They’ll look as if they work, and checking by using the thing, the method this book teaches, can’t tell you whether they’re safe.

A shopping list that adds onions wrong is easy to spot. A login with a security hole works perfectly for you and every honest user, right up until someone dishonest finds it.

So, firmly:

- **Payments.** Don't take real money through anything you built yourself. In a prototype, the checkout button goes nowhere. If your payment service offers a test or sandbox mode (check its own documentation), keep any checkout you try in that mode, where no real money moves. Get a developer's review before a single real payment.
- **Logins and accounts.** Don't build sign-in, passwords or password resets for real people. If a prototype needs a sign-in screen, make it a fake one that leads straight in.
- **Other people's personal data that you hold yourself.** A sign-up list kept by a form service is fine. Copying it into your project, or storing people's details in something you built, is level 4. For Pantry, food allergies and medical diets are health information, and a grocery budget linked to someone's bank account is financial information. Prototype with fake profiles, and don't collect the real thing without a developer and advice on the rules that apply.

You've reached level 4 if you're about to ask a real person for a password, a card number or a health detail, if strangers can create accounts, or if people's details are about to land in your own project. Stop there, and launch nothing until a developer has reviewed it. Chapter 13 covers money and personal data.

Scale is the other reason to stop. If many people will depend on a tool every day, decide who fixes it when it breaks, and have a developer review anything the business can't run without.

From prototype to MVP

When Pantry is ready to become an MVP, this is roughly what a developer adds. Use it as a checklist for your first conversation:

- **Accounts:** real sign-in, password resets and a way for people to delete their account.
- **A real database,** so each household's plans are saved and kept apart from everyone else's.
- **A security review:** someone who knows the code checks it for holes before strangers use it.
- **Payments,** through a payment service, tested before any real money moves.
- **Hosting** that runs the whole app, database and all.
- **Monitoring:** something that watches the live app and alerts a person when it breaks.

NOTE: Pantry is the book's example meal-planning app, and the levels are the "Who will rely on it?" ladder from Chapter 1.

CHAPTER 13 · EXCERPT

Safety, Privacy and Cost

By the end of this chapter you'll know which permission requests to approve and which to refuse, how to keep keys and other people's data out of harm's way, what to check before anything goes public or anyone else relies on it, and how to make your plan go further. It also gathers the ten mistakes this book has warned you about most.

In 2025, a user of an AI app-building service wrote that its AI agent had deleted his app's live database, the real data the app ran on, during a code freeze, when nothing was supposed to change. According to his posts, the agent then reported that a rollback couldn't restore the data, though he later wrote that the rollback did work. The company's chief executive called it unacceptable. The fix

wasn't a firmer instruction: the company launched separate development databases for building, apart from the live ones, and said the agent would ask before changing live data. Instructions guide Claude, but only settings decide what it can reach.

Permission requests: yes, no or “explain first”

In **Manual** mode, Claude asks before it changes a file or runs a command. The skill is spotting the few requests that need a second look.

Here are six common ones. The app words them its own way:

What Claude asks to do	What it means	Your answer
Create or edit a file in the project	It's doing the job you asked for	Yes
Open a page or a file you asked for	It wants to show or check its work	Yes
Install or download software	It needs a tool it doesn't have	Ask what and why first
Send your saved versions to GitHub	It's updating your on-line backup	Yes, if you set up the backup and nothing goes live from it automatically
Delete files	It wants to remove something	Only if you asked, after saving a version
Read or change something outside the project folder	It's reaching beyond this project	No, unless you asked

Take the same care with anything that sends something off your computer. Saying no is safe: Claude will stop, ask what you'd prefer or suggest another way. Judge any new request the same way, and if it does the same thing by a different route, say no again. When you're unsure, ask:

PROMPT

Before I approve that, explain in plain English what it does, what it will change, and whether it can be undone.

When a request comes out of nowhere

Watch for a sudden change of direction, especially after Claude reads a web page, a document or an email. That can be **prompt injection**: instructions hidden in content Claude reads, written to make it do something you didn't ask for. Anthropic builds in protections, but its documentation says no system blocks every attack. If something feels off, press `Esc`, then ask what prompted the action.

Limits that have to hold

One **Auto** default is easy to miss: it allows pushing (uploading saved versions) to any branch of your GitHub repository, including your main version.

If your site updates itself from GitHub, Auto could put a change live without asking. And a limit you state in the chat, such as "don't push," can drop out of a long session when Claude condenses older messages.

Project notes don't fix this, for the reason in this chapter's opening story. For a rule that has to hold, ask for a permission rule: a setting the app enforces, which makes Claude always ask before an action, or never take it, in every mode. For example:

PROMPT

Add a permission rule to this project's settings so you always ask me before pushing anything to GitHub, in every mode. Then explain in plain English what you added.

Ten mistakes to avoid

1. **Believing “done”.** Check it yourself against a known answer, awkward cases included (Chapter 5).
2. **Not saying how you'll check.** Put one known answer in every brief (Habit 1).
3. **Skipping the save.** Save a version before the next change (Habit 4).
4. **One endless session.** Start fresh (Habit 3).
5. **Correcting the same thing twice.** Ask Claude to put it in the project notes (Habit 2).
6. **Approving what you don't understand.** Ask Claude to explain first, and say no if in doubt.
7. **Too much freedom too soon.** **Accept edits** and **Auto** belong in a dedicated folder with saved versions and a backup.
8. **Keys and real data in the chat.** Keys go in a private file you fill in yourself; real data waits until fake data has proved the tool.
9. **Publishing without a check.** Look for keys, personal data and test leftovers, and know who can open every link.
10. **Building the risky parts alone.** Level 4 work, such as payments, logins or other people's personal data, needs a developer's review before launch.

45 Prompts to Copy

Ready-made requests for the jobs you'll do most often, grouped by job. Most follow Habit 1: they say what you want and how you'll know it worked. Replace anything in `<angle brackets>` before you send it.

They're starting points, not magic words. Add your own specifics, such as prices, columns or customers, and Claude has less to guess. A "Don't use it when" line marks when a prompt is risky or wrong.

TIP: If you paste the same prompt every week, ask Claude to turn it into a skill (prompt 33), so one short command does the whole job.

Starting a project

1. When you have an idea but no plan yet.

PROMPT

I want to build `<the idea in a sentence or two>`. I don't program. Before you build anything, interview me, one question at a time, about who it's for, what it must do and how I'll know it works. Then write a one-page plan in plain English and wait for my go-ahead.

2. When you want a plan before anything is built. Switch the mode selector to **Plan** first.

PROMPT

Plan this, but don't change any files yet: <describe the result>. List the steps in plain English, the files you'll create, anything you need from me and how you'll check it works. Point out anything risky, such as personal data, payments or something going public.

3. When you've just opened a new, empty project folder.

PROMPT

This folder is a new project: <what it is and who it's for>. Set it up so you can save versions, then create project notes (CLAUDE.md) covering what the project is, who it's for, how to check it and these rules: <your rules>. Show me the notes, then save a first version.

4. When you're not sure the idea is small enough.

PROMPT

Here's what I want: <describe it>. Is this one job or several? If it's too big to build and check in one sitting, split it into small steps, each ending with something I can try myself. Don't build anything yet.

5. When you come back to a project after a break.

PROMPT

Read the project notes, then explain this project in plain English: what it does, how I open or use it, and what changed in the last few saved versions. Tell me anything that looks unfinished. Don't change anything.

NOTE: That's the first group of the 45 prompts. The full appendix adds prompts for describing a design, checking work, fixing problems, data and spreadsheets, websites, automations, handing off to a developer, and safety and privacy.

THE FULL BOOK

Get the Full Book

This sample is a small slice of the book. The full edition has 13 chapters and five appendices, from setting up the desktop app to playbooks for founders, operators, creators and analysts.

What you get

- The complete book as a PDF, with clickable bookmarks, that reads well on a laptop, tablet or phone
- All 13 chapters: setup without a terminal, your first project, describing what you want, checking the work without reading code, project notes and saved versions, getting unstuck, four playbooks, skills, connectors, schedules and the cloud, and safety, privacy and cost
- Five appendices to keep open while you work: a plain-English glossary, 45 prompts to copy, four project notes templates, a troubleshooting guide and the sources behind the stories
- A 30-day refund if the book isn't for you

How to buy

1. Go to aiagentsforhumans.com/claude-code-for-non-coders and choose Get the book.
2. Pay through the secure checkout, run by Dodo Payments, which handles payment and any sales tax, VAT or GST.

3. Your download link arrives by email straight away, and you can download the book again any time from the Dodo Payments customer portal.

About the author

Pranav S. is an independent author. *Claude Code for Non-Coders* is a practical guide for founders, operators, creators and analysts who want Claude Code to build working tools for them, and who want to know how to check the results without reading code.

Its features and settings were checked against Anthropic's official documentation as of October 2026.

Claude Code for Non-Coders: Free Sample. A free sample of *Claude Code for Non-Coders: Build Websites, Tools and Automations Without Writing Code*, first edition, written for Claude Code as of October 2026. © 2026 Pranav S. You're welcome to share this sample with anyone, as long as it stays unchanged and free.

This is an independent, unofficial guide. It is not affiliated with, sponsored by or endorsed by Anthropic. Claude and Claude Code are trademarks of Anthropic, PBC. Claude Code changes often; where anything here differs from the official documentation, the documentation is the source of truth.