A CRASH COURSE · 13 CONCEPTS

Code You Never Write

Get AI to write, run, and fix the real, tedious work of your job — and trust the result — without writing or reading a single line of code.

You describe the problem **The AI writes & runs the code** **You verify the answer**

WHERE THIS STARTS

The bookkeeper's lost evening

Every month, two files were supposed to agree: what her company recorded spending, and what the bank actually charged. Every month she lost an evening hunting, line by line, for the few that didn't match.

Last month she typed the problem into an AI chat in plain English. A minute later: **23 mismatches** , each with its amount and date. The evening was gone.

WHAT'S MISSING FROM THE STORY

- ✕ She never learned to program
- ✕ She never read a line of the code
- ✓ **She described a problem and the work that ate her evening finished in the time it takes to pour a coffee**

THE WHOLE COURSE IN ONE LINE

You are not learning to write code. You are learning to be a good client for code.

Good clients don't lay bricks. They write clear briefs, check the work against things they *can* measure, and keep what they paid for.

THE PROMISE, STATED PRECISELY

Never write it. Never read its syntax. Always check what it did.

The freedom

You won't write code, and you never have to read its syntax — the punctuation and keywords that take months to learn. You won't even pick the language. The AI does all of that.

The responsibility

You will learn to check what the code *did* : confirm the totals, inspect the steps in plain English, catch the wrong answer. That's the whole job of a good client.

These two are not in tension. Not reading the syntax is the freedom; verifying the result is the responsibility.

TRY IT NOW · 2 MINUTES

Feel the gate swing open

Paste a few expense lines into any AI chat and ask it to total them. It doesn't answer from a glance - it **writes a small program, runs it on your numbers,** and reports back the exact totals.

- ✓ You chose no language
- ✓ You installed nothing, pressed no run button
- ✓ Then hand-check one slice — and watch it match

```
Here are my expenses for the month. Write and run
code to total them by category, find my biggest
category, and tell me the exact total. Show me
that the code actually ran.
Groceries 4,250 Fuel 3,100 Groceries 2,890
Internet 2,499 Fuel 2,750 Eating out 1,850 …
```

```
# then, the move that makes it a skill: Add up the
Groceries lines by hand: 4,250 + 2,890 + 3,120.
Does it match what your code reported?
```

PART ONE

The Deal

Three ideas that change what you believe you're allowed to ask for.

1 · Code isn't gated by coding

2 · What code actually is

3 · The dividing line

01 CODE IS NO LONGER GATED BY CODING

The AI is the developer. You are the client.

The developer works in seconds, for free, and never tires of your revisions. Three things matured together:

Writes

For one file, one folder, one repetitive task, modern AI writes *working* code on the first or second try.

Runs

It executes the code itself — in the chat's sandbox or on your machine — and sees the result. You copy nothing.

Repairs

When something breaks, the error goes back into the loop and the AI corrects its own course.

The unit of work changes. You no longer ask *"can I write this?"* You ask **"can I describe this?"** — a skill you already have.

FROM THE FIELD

Six professions, zero lines of code read

Accountant	Reconciled 1,400 rows in seconds; flagged 23 mismatches with amounts and dates.
Doctor	No-show rates by weekday and hour — Monday 9am ran 3× worse than average.
Marketer	Merged four platform exports into one table: spend, leads, cost-per-lead.
Teacher	Weighted finals, letter grades, and a one-line comment for 200 students.
Student	Renamed 300 scanned files using the date stored inside each photo.
Network engineer	Logs into 40 devices each morning, pulls status, writes a health report.

The pattern in every row: a problem that was always *describable* but never *commissionable* — until now.

02 WHAT CODE ACTUALLY IS

A list of exact instructions, followed perfectly

"Open the file. For each row, read the category and amount. Add it to that category's total. Print every total." In English, that's a procedure. In a language, it's code — run a million times without one slip of attention.

```
totals = {} for row in rows:
totals[row.category] = totals.get(row.category,
0) + row.amount
```

Did your eyes slide right off that? Perfectly fine — yet you can half-read it: "for each row, add the amount to that category's total." That's all the reading this course asks.

You're allowed to glance at it. You're never *required* to.

02 THE AI PICKS THE LANGUAGE

The choice follows the output — and it isn't yours to make

Python

When you want a **number, a cleaned file, a report, a renamed folder.**
The default for data work — and the language the AI knows best. You'll see it almost every time in this course.

JavaScript

When you want a **thing someone clicks** — a web tool, a game, a shareable calculator.
That was the playful course's world. This course barely visits it.

A good client describes the building and lets the architect choose steel or timber. **You describe the problem; the AI decides the language** — and it's right about that far more reliably than a beginner would be.

THE ONE PROPERTY WORTH MEMORIZING

Code is exact — in both directions

It will be exactly right

It computes your totals to the last cent, with zero arithmetic errors, at any scale.

…and exactly wrong

A script with a misunderstanding renames 300 files *wrongly* just as fast as it would rename them rightly.

A recipe is code for a human. "Simmer 20 minutes" makes dinner; "simmer 20 hours," obeyed perfectly, makes charcoal. The robot didn't fail — the brief did. That's why **verification** and **blast radius** exist.

03 THE DIVIDING LINE

A question for the mind, or a job for the hands?

This is the one decision you make *before* any prompt. AI has two ways to help — and most people only ever use the first, then wonder why the "analysis" of their spreadsheet had wrong totals.

Answer

Draft, advise, summarize, explain, brainstorm — from its own reasoning. It skims, like a human.

Compute

Write and run code that operates on your *actual* data. It processes, like a machine counts.

Wrong totals happen when the AI **answers when it should have computed** — describing your data from a glance instead of processing it.

When it trips one of these, it's a code problem

V

Volume

More items than you'd do by hand. 300 files, 5,000 rows, 80 PDFs.

P

Precision

A wrong digit has consequences. Money, payroll, grades, dosages.

R

Repetition

You'll face it again next week. The script becomes an asset.

F

Files

It lives in files, not sentences. Spreadsheets, exports, folders, logs.

Trip none of the four → it's an **answer problem** .

Trip even one → it's a **code problem**.

RUN YOUR WEEK THROUGH THE FILTER

Sort the task before you prompt

"Draft a polite email declining a meeting."	Answer	No volume, precision, or files.
"Which of these 80 contracts have a non-standard clause?"	Code	Volume + Files.
"What did I spend on fuel this year?"	Code	Precision + Files.
"Is my business idea any good?"	Answer	Judgment.
"Every Monday I combine three exports into one report."	Code	Repetition — the strongest signal.

The trap: "Roughly how did sales trend?" feels conversational — but if a decision rests on the number, it's a code problem no matter how casually you'd phrase it.

PART TWO

Commissioning Code

The full client-side craft: make code run, brief it well, verify it, survive failure, and keep it.

4 · Make the hands move | 5 · Brief it | 6 · Verify | 7 · When it breaks | 8 · Keep it

04 MAKE THE AI'S HANDS MOVE

An estimate is formatted exactly like a computation

Same confident tone, same tidy numbers. On anything that *sounds* answerable from a glance, the AI may skip the code and guess. This is the silent failure mode — enemy number one.

"WHAT DOES THIS DATA SHOW?"

Glance-based summary. Estimates dressed as facts.

"WRITE AND RUN CODE TO COMPUTE IT."

Code, every time. Risk: minimal.

The habit: **never leave the decision to the AI on anything that trips Precision.** Say the words — and notice you say "code," never "Python."

PIN THIS ABOVE YOUR DESK

The three-line incantation

```
1 · Write and run code to answer this.
2 · Show me the code you ran.
3 · First, tell me the exact row count, the column names, and
the date range of the file.
```

LINE 1

Forces computation.

LINE 2

zroof: no code block, no code ran — whatever the prose claims.

LINE 3

The cheapest lie detector: real reading gets these exactly right.

05 BRIEF THE PROBLEM, NOT THE PROGRAM

The best briefs contain no technical language at all

You don't specify loops, libraries, or even the language. You specify what a good outcome looks like — exactly as you would for a human assistant — and the AI translates intent into implementation.

People who half-know programming often write *worse* briefs than total beginners: they micromanage the how and under-specify the what.

Don't know your own edge cases?

Make discovery the first prompt:

```
"Before doing anything, examine the files and
tell me what could be ambiguous, inconsistent,
or surprising in here."
```

Inspect, ask, *then* brief — the brainstorm-iterate loop, applied to data.

THE FIVE-SECTION BRIEF

A complete brief answers five questions

Goal

What problem is solved when this works?

Input

What am I giving you? Files, format, rough size.

Output

What exactly do I want back? A number, table, report.

Rules ★

The constraints a stranger wouldn't know.

Edge cases ★

What should imperfect data do?

It's a Markdown spec — every heading is a thing the AI no longer has to guess. The two starred sections are the ones beginners skip, and the ones that prevent every wrong answer.

WHERE THE BRIEF IS WON

Every wrong analysis traces to a rule you knew and didn't state

Rules

Where your professional knowledge lives. The AI knows Python; it does *not* know your fiscal year starts in July, that "Pending" means unpaid, or that the Lagos branch was sold in

The school admin's missed families = a missing rule

Edge cases

Decide in advance what imperfect data should do — because your data *is* imperfect. One sentence does it:

"Hit a row you can't read? Don't guess — skip it, and list every skipped row at the end."

That one sentence converts **silent corruption** into a **visible report**.

06 VERIFYING WORK YOU CANNOT READ

If you can't read the code, how do you know it's right?

The same way you check any expert whose craft you can't audit.

You don't re-derive tax law to check your accountant

You check whether the wall is straight and the door closes

You verify **outputs against independent knowledge** — and you, the domain expert, have knowledge the AI never will.

THE VERIFICATION LADDER · RISING WITH STAKES

Five checks. The first three cover most days.

1	**Known-answer test**	Run it on one slice whose answer you already know. The single most powerful move.
2	**Reality questions**	Rows in vs. rows out. Totals plausible. Biggest item the one you know.
3	**Plain-English replay**	"Explain what the code did." Wrong logic reads wrong in English too.
4	**Adversarial pass**	"Find any way this is wrong. Score your confidence 1–10."
5	**Cross-model check**	Take the brief to a second AI family and compare the numbers.

Never skip the one: never act on a precision-critical number you haven't tested against an answer you independently know.

STORY · THE KNOWN-ANSWER TEST, ELEGANTLY

The pharmacist who salted her own data

Before trusting a script to cross-check her dispensing log against stock counts, she planted a trap: one fake discrepancy she knew about because she created it.

The script caught her plant — **plus four real discrepancies she didn't know about.** The plant is what let her believe the four.

90sec

Salt your data with one known error and see if the code finds it.

A known-answer test in its most elegant form — and it costs ninety seconds.

07 WHEN IT BREAKS

An error isn't the project failing — it's the computer saying what it needs

And you have an expert on staff, already in the conversation, who reads that language fluently. The entire skill, for red-text errors:

```
It stopped and showed this error. Diagnose it, fix the code,
and run it again: [paste the red text — all of it]
```

The marketer & the "codec": she pasted an error she didn't understand a syllable of. The AI fixed it in 40 seconds. She never learned what a codec is — and never needed to. **Fluency in errors is not required. Willingness to paste them is.**

THREE PATTERNS COVER NEARLY EVERY BREAKDOWN

Diagnose by symptom, not by panic

Red text, script stopped	Paste the full error: "diagnose, fix, rerun." Most errors die on the first paste.
Runs, but a number is off	State the symptom + your expected value. Ask it to show row counts and sample rows *before* fixing.
Same fix fails 3× in a row	Stop digging. "Step back, restate the problem fresh, propose two completely different approaches." Three strikes means the **approach** is wrong, not the typing.

If the chat has gotten long and confused, start a clean one with the brief and the lesson learned — that's context rot, and abandoning beats rescuing.

08 KEEP THE SCRIPT

A solved problem becomes a button you press forever

May's reconciliation took effort. In June, the same job is one sentence: *"Run my reconciliation script on these two new files."* The marginal cost of every future month rounds to zero.

This is the moment you stop *using* AI and start **accumulating** with it.

① Ask for the script as a **named file** with a plain-English note at the top

② Keep the **brief beside the script** — the what & why, for future edits

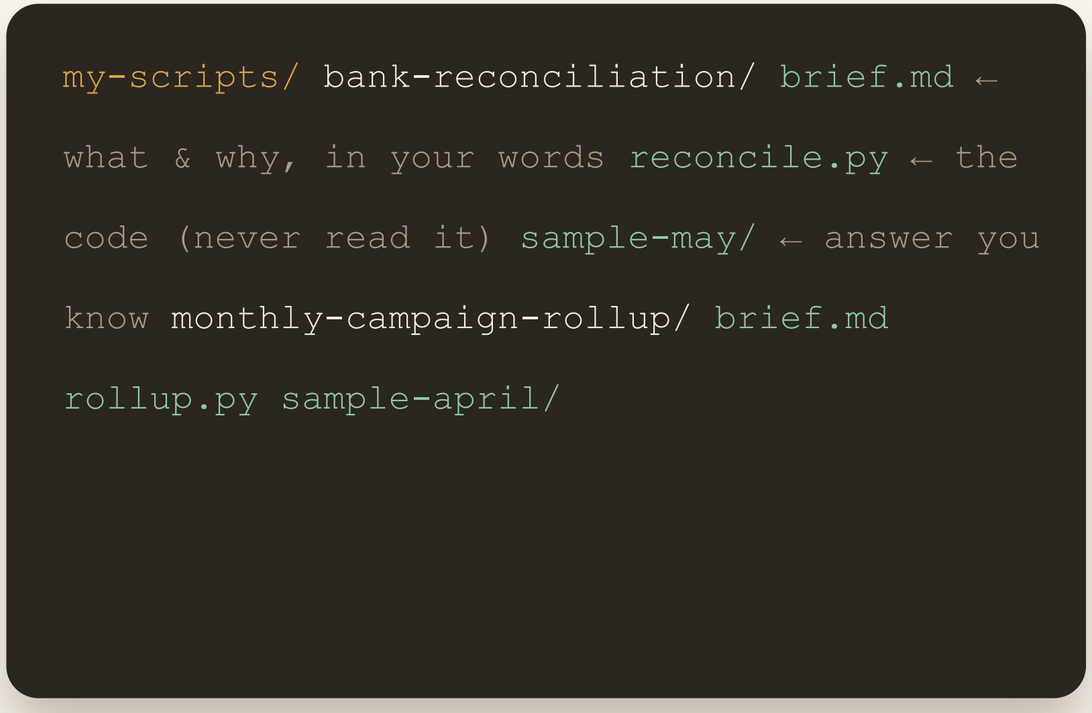
③ Give them a **home** : one folder, with a known-answer sample inside

DESCRIBE ONCE · RUN FOREVER · HAND TO ANYONE

One folder per task is your first owned asset

A clinic manager built her no-show analysis in March and kept the script. By August it was a 30-second Friday ritual — and when a new branch opened, she handed over the folder and its manager ran it on day one, no questions.

One afternoon of describing, six people's recurring work automated.



PART THREE

One Problem, Five Surfaces

Everything you've learned works everywhere. What changes is *where the code runs and what it can touch.*

The running job: a folder of twelve monthly expense files. Merge them, total by category, flag duplicates, and produce a one-page report with a chart.

09 THE HOME SURFACE · BROWSER CHAT

Code runs in a sandbox — a temporary computer on their side

Your uploads go in; results, files, and charts come out; nothing on *your* computer is touched. The zero-risk surface to learn on. Claude.ai, ChatGPT, and Gemini all work this way.

Best at

Zero install, zero risk. Perfect for one-off and exploratory jobs. The full brief → code → verify → iterate loop, all visible.

Where it ends

It sees only what you upload — 40 files means 40 uploads. The sandbox is temporary, so scripts must be downloaded or they die with the chat.

The chat sandbox is a **workshop you visit** — not where your files live.

10 CLAUDE CODE & OPENCODE

The terminal is just a chat box that sits inside a folder

If the word makes you flinch, hold that reframe. You type sentences; the agent answers. The difference from the browser is everything around the chat:

Sees files

Already there. No uploading.

Runs directly

On those files, on your machine.

Scripts persist

Saved forever. Rerun next year.

Self-heals

Fixes its own errors in a tight loop.

This is the surface for the most demanding jobs — code that must run **on a schedule** , touching systems a browser tab can't reach. The network engineer's 40 devices, every morning, before he reaches his desk.

COWORK & OPENWORK

Verification, built into the surface

Desktop apps with the same file-touching power as the terminal — wrapped in a normal window, built for knowledge workers. Underneath, they're very often writing and running code. Same Python, different costume.

Their defining feature: a built-in rhythm of **plan first, act after approval.**

1 · It plans

"I'll read the 12 files, merge on these columns, flag duplicates without deleting, produce report.html…"

2 · You read it — in plain English

Catch the wrong assumption while it's still words, not actions.

3 · You approve

The plain-English replay, moved *before* the work instead of after.

CHOOSING A SURFACE · ONE GRADUATION PATH

Stay on Claude.ai until uploading becomes the annoying part of your week

One-off question about a file or two	**Claude.ai**	Upload, compute, verify, done. Nothing at risk.
Lives in your folders; you'll repeat it	**Claude Code / OpenCode**	No uploads, scripts stay, errors self-heal.
Want an app + a plan to approve	**Cowork / OpenWork**	Plan-then-approve is verification built in.
Must run on a schedule / touch other systems	**Claude Code / OpenCode**	Only code on your machine runs while you sleep.

Chat is where scripts are born; your machine is where they live. In each pair, the first is commercial, the second open-source.

PART FOUR

Power, Safely Held

The safety rules for code that touches real things — and an honest map of where one-prompt code ends.

Deleted files often skip the bin. Edited files keep no undo.

1 Work on copies

"Copy the folder to a backup first, then work only on the original." Until a script has earned trust.

2 Demand a dry run

"Don't change anything yet. Show me every operation you'd perform, and wait for approval."

3 Scope the territory

Point at the smallest folder that holds the problem. Never your whole disk.

4 Output to new files

"Write results to a new file; leave the originals untouched." Every run reversible by deletion.

Three sentences per brief, ninety seconds per run. Trust is granted to **specific scripts with track records** — not to the technology.

13 THE EDGE OF THE MAP

Know exactly where the territory ends

Software others log into

Accounts, payments, uptime — products, not scripts.

Always-on automation

Runs unattended, where your checks can't reach.

High-stakes, no undo

Let code *prepare* the tax filing; let a human *fire* it.

Judgment in disguise

"Which employees to promote" — code computes, you decide.

Inside the territory, yours today: every reconciliation, rollup, rename, merge, clean, cross-check, flag, chart, and report that trips Volume, Precision, Repetition, or Files. For most professionals, that's hundreds of hours a year.

TAKE THE TRAINING WHEELS OFF

Four Projects

Each chains all thirteen concepts together — and ends with something real. Do Project 1 this week.

30–90 MINUTES EACH · ON A FREE ACCOUNT

From rehearsal to real, on your own

1 · The Year of You — 45–60 min

Export a year of your own data. Brief it, force code, verify with a known-answer test, finish with a report and a saved script.

2 · The Folder You've Avoided — 30–45 min

A volume problem with the full safety ritual: copy first, demand the dry run, read it, catch one operation to change, approve.

3 · The Handover — 60–90 min

Turn a recurring task into an asset a colleague can run — using only what's in the folder, with you out of the room.

4 · One Problem, Two Surfaces — 45 min

Run Project 1 a second way — a second AI family, or your own folder — and feel the surface boundary for yourself.

THE WHOLE COURSE · ONE LINE EACH

The thirteen concepts

1 Code is no longer gated by coding.

2 Code is exact; the AI picks the language.

3 Four signals: Volume, Precision, Repetition, Files.

4 Say the words: write and run code; show it ran.

5 Brief the problem: Goal, Input, Output, Rules, Edge cases.

6 Verify against what you independently know.

7 Errors are dialogue — paste the red text.

8 Keep the script; describe once, run forever.

9 Claude.ai runs code in a zero-risk sandbox.

10 Claude Code / OpenCode: the agent in your folder.

11 Cowork / OpenWork: plan-then-approve.

12 Blast radius: copies, dry runs, scope, new files.

13 The edge of the map needs the rest of the book.

ONE IDENTITY · ONE SKILL

You are the client, not the contractor

And the new skill the prompting course never needed: you can **trust work you cannot see** — because you tested it against something you already knew.

None of them became programmers. They became good clients. Now so are you.