FOR EVERYONE · STUDENTS TO PROFESSIONALS

General Agents for *Everyone*

Let AI Agent work directly on your computer: you describe what you want, they do the work, you review the result. Fifteen concepts that cover about 80% of what you'll use day to day.

From Answering to Acting

A CHATBOT

Answers the question.

It tells you what you *could* do. You still open every file and make each change by hand.

AN AGENT

Does the work.

It reads your files, makes the edits, runs the commands — and keeps going until the task is done.

You describe what you want. It does the work. You review the result — the same loop, whether the task is code or notes or a spreadsheet.

★ HOLD ONTO THIS

Give the model the *right information* at the *right time* — and keep everything else out.

Nearly every ”advanced” technique in this course is just a version of that one sentence. When something feels complicated, come back here.

Two Tools, One Set of Skills

	CLAUDE CODE	OPENCODE
made by	Anthropic	Open-source community
runs on	Claude models	Claude, GPT, Gemini, local — any model
best at	Top Claude performance, out of the box	Flexibility, cost control, free models
cost	Subscription or API	Free models, or bring your own key

If a technique works in **both**, it's a real skill — not a tool-specific trick. *Everything in this course works in both.*

The Course Map — 8 Parts, 15 Concepts

Part 1 · C1-4
Foundations
What they are, plan mode, permissions, choosing a model

Part 2 · C5-7
Context Management
Why chats get worse over time — and how to fix it

Part 3 · C8
The Rules File
Permanent instructions your project reads each session

Part 4 · C9-11
Personalizing
Commands & skills, hooks/plugins, subagents

Part 5 · C12
Connecting Out
Linking the model to real services with MCP

Part 6 · example
Worked Example
One full task, start to finish, no code required

Part 7 · C13
Where to Run
Terminal, IDE, web, desktop — and where to start

Part 8 · C14-15
Both Together
A personal library, memory, two tools in concert

Read straight through, or jump to Part 6 to learn by doing first — then circle back. *The throughline ties it all together.*

01

Foundations

The four ideas everything else is built on.

CONCEPT 01

What These Tools Actually Are

The whole mindset, in three words: *stop asking, instruct.*

ASKING A QUESTION · WEAK

"How do I organize my notes?"

You get an explanation you still have to act on yourself.

GIVING AN INSTRUCTION · STRONG

"Read every file in [redacted]. Make one summary of every action item by person. Skip anything marked private."

You get the work done — inside clear boundaries.

1

CONCEPT 02 · THE MOST UNDERUSED FEATURE

Plan Before You Build

Put the model in a *look-but-don't-touch* state. It reads, thinks, and writes out a plan — before it changes a single thing.

CLAUDE CODE

Shift+Tab

OPENCODE

Tab

You catch a wrong assumption in the plan, not in a finished draft. And the model simply does better work when it's made to think first.

Rule of thumb — more than ten minutes of work? Plan first. Like outlining an essay before you write it.

2

CONCEPT 03

Keep Your Hand on the Wheel

Every action asks first. When you're starting out — *keep it that way.* Watch what it does before it does it.

SAFE TO AUTO-APPROVE

```
# reading is always safe
read · edit · write
npm test · npm run lint
git status · git diff
```

NEVER AUTOMATIC

```
# destructive — keep the gate up
rm -rf *
git push *
npm publish *
```

After a few sessions, auto-approve the boring-safe actions. Keep the dangerous ones manual — forever.

CONCEPT 04

Match the Model to the Task

PLAN & REASON

A strong model.

The hard part is figuring out *what* to do. Spend your best model here.

EXECUTE

A cheap or free one.

Once the plan is clear, the doing is mechanical. Let a low-cost model grind it out.

Switch in one command — `/model` in Claude Code, `/models` in OpenCode.
Most beginners never touch this. That's a mistake in both directions.

4

02

Context Management

The single most important idea in the course.

05 Context rot is real

06 Reset, or compress

07 Pick up where you left off

\+ Diagnostics: when context goes bad

CONCEPT 05

Context Is the Whole Game

Think of it like a group chat. The model only sees the messages in *this* conversation.

There's a ceiling on how much it can hold — the **context window.** As a chat grows long and messy, the model loses track and starts to drift. That drift is **context rot.**

Glance at the gauge — don't wait for it to fill.

Quality slips well before the window is technically full, and a bloated chat also costs more on every message.

5

CONCEPT 06 · TWO COMMANDS, DON'T MIX THEM UP

Reset, or Compress

`/CLEAR`

Start fresh.

Erases the whole conversation. Reach for it when you switch to a different task.

`/COMPACT`

Same task, less clutter.

Keeps the important parts, drops the rest. Reach for it when the same task drags on.

Don’t confuse them. Run `/clear` in the middle of a task and you lose all your progress.

6

CONCEPT 07

Every conversation is saved. Close the tool, shut the laptop, come back tomorrow — and *continue exactly where you stopped.*

Resume with `claude --resume` or `/sessions`. For big jobs, also save the plan to a file — a backup that survives anything.

7

Signs the Context Has Gone Bad

- → It apologizes, but makes no progress
- → It rewrites the same code in circles
- → It invents files or variables that don’t exist
- → It contradicts limits you set earlier

THE PRESCRIPTION

Stop typing. Don’t fight it with another prompt.

Run `/compact` or `/clear` and reset.

Five minutes resetting beats an hour arguing with an AI that's lost the thread — and the messy chat was costing you more, too.

PART THREE — THE RULES FILE

One File the Model Reads Every Time

Permanent instructions for your project. *Keep it short* — write only what the model can't figure out by looking.

CLAUDE.md · AGENTS.md

The conventions, the landmines, the rules. Not the things it can read for itself — those just crowd the file and waste the context it buys you.

```
# Project: my-app

## Stack
Next.js · TypeScript · Postgres

## Critical rules
- Never edit src/generated/ (codegen)
- All routes use the auth middleware
```

8

04

Personalizing Your Tool

Three ways to keep the model focused and your chats clean.

CONCEPT 09

Commands & Skills

Two names for one idea: a *saved, reusable instruction* in a file, so you never retype it. They differ only in who invokes them.

COMMAND

You invoke it.

Type `/name` and the saved prompt runs. You decide exactly when.

SKILL

The model invokes it.

You write a description; the model reaches for it on its own when a task matches.

Both live as plain files in your project — `.claude/skills/` — so they travel with the code. You don't even have to write them from scratch; each tool can generate one for you.

9

CONCEPT 10 · HOOKS · PLUGINS

Rules That Always Run

For what must be true *100% of the time.* The model can't forget them, because it never gets a choice.

SKILLS ARE FOR —

what you'd like the model to remember *most of the time.*

HOOKS & PLUGINS ARE FOR —

what has to be true *every single time.*

Auto-format on every save. Block a forbidden command, always. The guarantees you can't leave to memory.

10

CONCEPT 11

Subagents

Send a helper to do the *heavy reading* in a separate room.

A subagent has its own context window. You hand it a task; it digs through files in private; it returns just a summary. All that searching and log-dumping never clutters your main conversation.

"Investigate why the build is slow and report back" — perfect subagent work.

11

PART FIVE — CONNECTING TO THE WORLD

MCP, Used Honestly

A standard way to plug the model into apps you already use — *Slack, GitHub, Notion, databases.*

WHAT IT UNLOCKS

The model can read from and act inside those services — pull an issue, post a message, query a table — with no copy-pasting in between.

THE DISCIPLINE

Add only what you actually need. Every connection is more for the model to wade through. A lean setup stays a sharp one.

12

06

A Complete Worked Example

One full task, start to finish. No coding required.

NO CODING REQUIRED

Messy Notes In, One Rollup Out

Five meeting-note files — each labels its to-dos differently (`Action Items` , `Todos` , `Next Steps` , `todo`) — become *one clean weekly rollup, grouped by person.*

THE CATCH

One file marks items `[private]` — those must never make it into the summary. A rule you'll set once, and trust.

IF YOU'VE EVER TAKEN NOTES

…in a meeting, a class, or a group project, you already understand this task. That's the whole point.

Eight Steps, Every One a Concept in Action

1 **Write a short rules file** — tell the model where the notes live and what the headings mean.

2 **Enter plan mode** and describe the goal in one breath.

3 **Read the plan,** catch the wrong assumption, correct it.

4 **Approve** — let it gather every action item across all five files.

5 **The private items stay out** — the rule held without you watching.

6 **Group by person,** review the draft rollup.

7 **Save the plan to a file** so next week is one command.

8 **One clean summary** — written for you, not just suggested.

Done once in Claude Code, once in OpenCode — the steps are identical. That's the proof these are skills, not tricks.

★ WHAT JUST HAPPENED

No code. No APIs. Just eight small acts of *managing the model's attention.*

Right information, right time, everything else kept out — the throughline, end to end. You've now seen the whole skill on a real task.

PART SEVEN — WHERE YOU RUN THESE TOOLS

Terminal, IDE, Web, or Desktop

01
Terminal
See every file read and command run.

02
IDE plugin
Right beside the code you're editing.

03
Web
From anywhere, nothing installed.

04
Desktop
A windowed home for longer work.

13

Start in the terminal or an IDE.

Once you can see what's happening underneath, move wherever's convenient.

08

Both Tools, Together

Grow your setup slowly — then let the two tools play off each other.

CONCEPTS 14 & 15

Grow Your Setup Slowly

14 · PERSONAL LIBRARY

Stop copy-pasting the same rules into every project. Promote the ones you reuse to a personal folder that every project inherits — `@~/.claude/style/…`

15 · MEMORY

Resuming chats plus a good rules file is enough for most people. Want more? Add a `notes/` folder the model can write to and read back.

Don’t build it all in a weekend. Add one piece at a time, only when a real problem asks for it.

Two Tools, Working Together

PLAN / EXECUTE SPLIT

Plan with a frontier model; hand the plan file to a cheap session to carry out. The plan is the contract — it survives even if a session is lost.

CROSS-MODEL REVIEW

Have one model check another's work. A different provider catches different mistakes — Claude reviewing Claude misses the same things twice.

Work on separate copies with git worktrees so the two never overwrite each other. Small task? One tool is plenty.

The 15 Concepts at a Glance

01 **They act** — give instructions, not questions.

02 **Plan mode** — see the plan before anything changes.

03 **Permissions** — approve by hand at first.

04 **Match the model** — strong to plan, cheap to execute.

05 **Context rot** — long chats drift and cost more.

06 **Clear vs compact** — fresh start vs. less clutter.

07 **Resume** — pick up tomorrow where you stopped.

08 **Rules file** — short; only what can't be inferred.

09 **Commands & skills** — saved, reusable instructions.

10 **Hooks & plugins** — rules that run every time.

11 **Subagents** — delegate heavy reading off to the side.

12 **MCP** — connect real services; add only what you need.

13 **Where to run** — start in the terminal or IDE.

14 **Personal library** — share your setup; grow it slowly.

15 **Memory** — a rules file is enough; add notes if needed.

Reading this won’t make you good. *Using it will.*

Keep a terminal open. Try each concept on a real task. The skill is in the reps — and it transfers between both tools.