

Lesson 03 — Ways to use AI

Chat · API · CLI · Code & Agents — four doors into the same building.

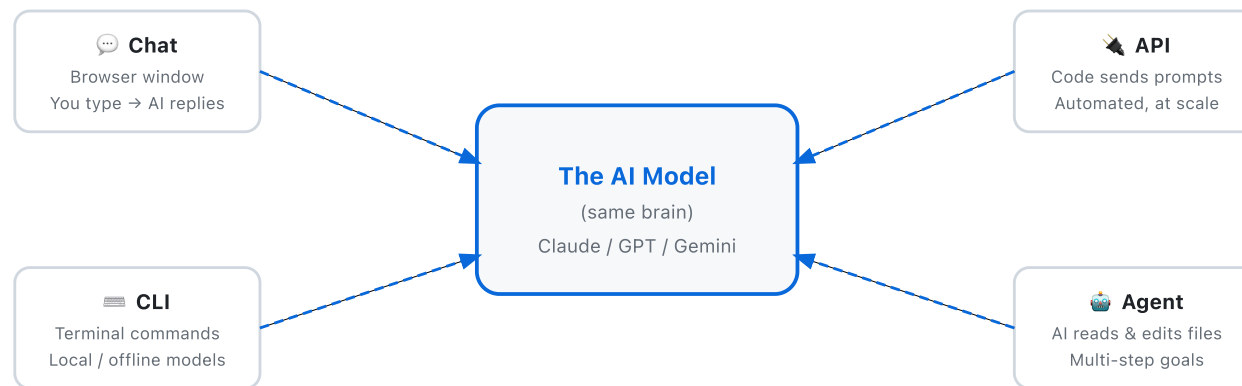
AI for Dummies

Module 3

The AI model doesn't change. *How you reach it* does — and that choice changes everything about what you can accomplish.

Four doorways — one building

All four paths reach the same AI model. The door you pick determines *who drives*.



Same expert. Same intelligence. The door you pick determines **who drives**: you by hand, your code automatically, a local tool privately, or an agent acting on your behalf.

The four doorways — click to explore



Chat

No skill needed



API

A little coding



CLI

Comfort with commands



Code / Agent

Technical comfort

👉 Click a card above to see details

Doorway 1 — Chat (the front door)

What Claude.ai looks like:

Claude.ai — New conversation

Summarize this email in 2 lines.

The email asks to reschedule Friday's meeting to next Monday. No action needed until you reply.

Type a message...

What you can do right now:

- Ask any question, get an explanation
- Draft emails, reports, or social posts
- Paste a document → instant summary
- Debug a snippet of code
- Brainstorm, translate, learn anything

The limit: one exchange at a time. Repeat the same thing 300 times? That's the API's job, not Chat's.

Rule: Start here. Stay here until Chat can't do what you need. Most people never need to leave.

Chat in action — a real 3-turn conversation

Watch how a follow-up message refines the answer without starting over.

You

Claude

▶ Replay

The Chat superpower: you refine, you don't restart. "Shorter," "more formal," "add examples" — each follow-up builds on context the model already has.

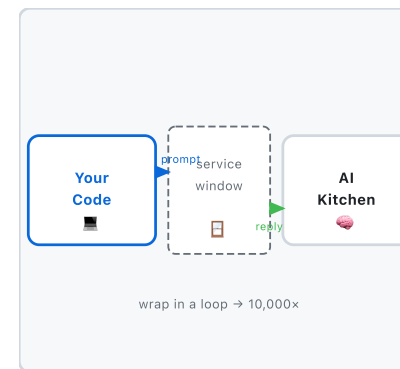
Doorway 2 — the API (kitchen window)

API = **Application Programming Interface**. Instead of *you* typing, *your code* sends the message and handles the reply — automatically, in a loop, at any scale.

Chat = dining room. You sit down, the waiter takes one order, brings it back.

API = kitchen window. You slide structured tickets through. Thousands of tickets. Structured responses back. No waiter needed.

Same chef, same food — different doorway.



When you actually need the API



You have 800 customer reviews and want a one-sentence sentiment tag on each.

Copy-pasting 800 times isn't a plan. A 10-line Python script that loops over your spreadsheet: that's the API.



You're building an app with an AI-powered feature.

The app calls the API on behalf of every user. That's how every AI-powered product in the world is built.



You want AI to run on a schedule — every morning, process last night's data automatically.

Script + scheduler + API = fully automated pipeline. No human clicking required.

Rule of thumb: if you'd need to type the same prompt more than a handful of times, the API removes all the repetition.

API in code — the canonical curl call

This is the actual Anthropic API call, straight from the documentation. Every library wraps exactly this.

▶ Replay

```
curl https://api.anthropic.com/v1/messages \
-H "x-api-key: $ANTHROPIC_API_KEY" \
-H "anthropic-version: 2023-06-01" \
-H "content-type: application/json" \
-d '{
  "model": "claude-sonnet-4-6",
  "max_tokens": 1024,
  "messages": [{"role": "user", "content": "Explain what an API is in one sentence."}]
}'
```

The Python equivalent — 7 lines

Install once, then call Claude from any Python script. This is a *complete*, working program.

```
# Install the library (one-time)
pip install anthropic

import anthropic

client = anthropic.Anthropic()
# reads ANTHROPIC_API_KEY from env

message = client.messages.create(
    model="claude-sonnet-4-6",
    max_tokens=1024,
    messages=[{
        "role": "user",
        "content": "Write a 2-line welcome email."
    }]
)

print(message.content[0].text)
```

What each part does:

- `Anthropic()` — connects using your env key
- `model=` — pick Haiku (cheap), Sonnet, or Opus
- `max_tokens` — cap the length of the reply
- `messages=` — the conversation so far
- `.content[0].text` — the text of the reply

Wrap `client.messages.create()` in a `for row in csv:` loop and you've just automated your entire spreadsheet.

The API key — your password *and* credit card

An API key is a long secret string. Every call made with it is **billed to your account** — even if someone else makes the call.

🚫 Never do this

- Paste it into a chat window
- Hardcode it directly in your script
- Commit it to a GitHub repo
- Share it in a message or email
- Leave it in a screenshot

✅ Always do this

- Store it in an **environment variable**
- Set a **monthly spending cap** immediately
- Use a separate key per project
- Rotate it the moment you suspect a leak
- In code: `os.environ["ANTHROPIC_API_KEY"]`

Leaked keys are used by strangers and run up *real* bills. It happens to beginners constantly — paste key into tutorial, push to GitHub, get charged \$300. Set the spending limit first. It's the one safety net that stops the damage.

Doorway 3 — CLI (the terminal)

The terminal is a text window where you type commands. It looks scary. It's just... typing.

```
$ ollama run llama3.2
pulling manifest... done

>>>
What is the capital of France?
The capital of France is Paris. It has
been the country's capital since 987 AD.

>>>
/bye
```

That's a real CLI session. Fully offline, zero cost after download, 100% private.

CLI is great for: local AI tools (Ollama, LM Studio CLI), running scripts, servers with no GUI, developer automation workflows.

Useful Ollama commands:

```
ollama list          # installed models
ollama pull mistral  # download model
ollama serve         # run as a server
```

Gentle CLI intro comes in **Lesson 04** when we install Ollama. You don't need to touch it today.

Doorway 4 — Agents (AI that *acts*)

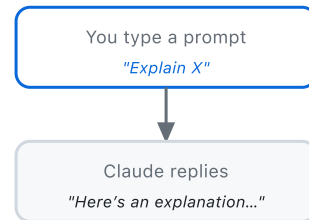
Chat answers. An agent *executes*. Give it a goal and step through how it works.

Your instruction: "Fix the bug in auth.py and make sure the tests still pass."

Next step ▶

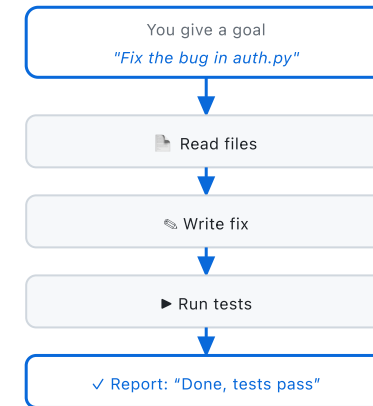
Agent vs Chat — the structural difference

CHAT — you steer every step



*You decide what to do next.
Every step needs you.*

AGENT — you set the goal



You review and approve.

Key difference: Chat gives you advice. An agent *takes action*. That power comes with a responsibility: always review what the agent proposes before it changes real files.

Agent example — Claude Code session

Claude Code is Anthropic's coding agent. It runs in your terminal, reads your project, and edits files. Here's what a session looks like:

```
# Inside your project folder:

$ claude
✓ Connected to Claude (claude-sonnet-4-6)

> Rename all Python variables called tmp to temp_buffer in the project.

🔍 Scanning project... found 6 files with tmp

📄 Proposed changes:
+ utils.py line 14: temp_buffer = []
+ parser.py line 88: temp_buffer = data
... 4 more changes

> Looks good, apply all changes.

✓ Applied 6 changes across 6 files. No tests broken.
```

You need: a paid Claude plan or API key. Install:

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

 or

```
curl -fsSL https://claude.ai/install.sh |
```

```
bash
```

Safety rule: always run Claude Code on version-controlled code. Read what it proposes *before* approving. It can and will change real files.

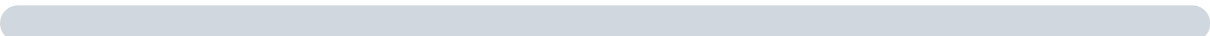
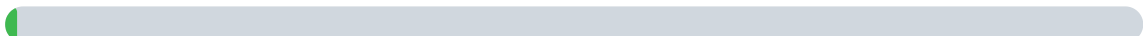
All four doorways compared

Doorway	Skill needed	Install?	Best for	Cost	Privacy
 Chat	None	No	Questions, writing, learning	Free tier	Cloud
 API	A little coding	pip install	Automation, apps, batches	Per token	Cloud
 CLI	Typing commands	Yes (Ollama)	Local tools, scripts, servers	Free	Local
 Agent	Tech-comfortable	Yes	Coding, multi-step file tasks	Per token	Cloud

Pick the doorway by what you *need*, not by what sounds most impressive. Chat solves roughly 80% of everyday problems and needs zero setup.

Cloud vs local — cost at scale

Cloud API costs money per token. Local models (Ollama) are free per call after setup. Here's what 10,000 short calls looks like:

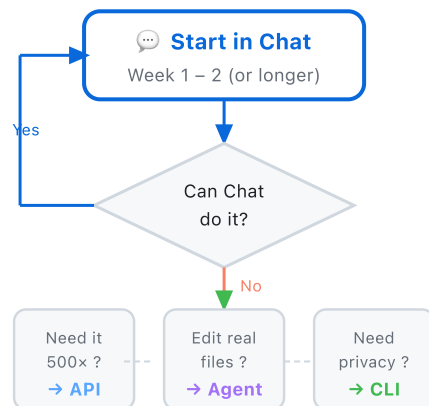
Claude Haiku (cloud)		~\$0.75
Claude Sonnet (cloud)		~\$3.00
Claude Opus (cloud)		~\$15.00
Local (Ollama / LM Studio)		\$0.00

~50 input + 50 output tokens per call. Prices illustrative — check provider docs for current rates.

Cloud: best quality, easiest setup, pay per use.
Data leaves your machine. Ideal for most tasks.

Local: free per call, fully private, works offline.
Needs decent hardware (16 GB RAM recommended). Perfect for sensitive data or high volume.

The beginner path — start at Chat, branch when needed



No prize for using a fancier door than you need.

Week 1–2: Chat only. Use Claude.ai for real daily tasks. Learn how to ask well, how to follow up, and when to verify answers.

 **"I need this 500 times"**
→ Move to **API**

 **"I want it to edit my files"**
→ Move to **Code / Agents**

 **"I need total privacy / offline"**

→ Install [Ollama](#) & use the [CLI](#)

Lesson 03 — key takeaways

- **Chat** — browser window, zero setup, start here. Handles ~80% of everyday AI needs.
- **API** — your code sends the prompt automatically; the doorway for automation, apps, and batches.
- **CLI** — the terminal; type `ollama run llama3.2` for a local, free, private AI session.
- **Agents** — AI that plans multi-step actions on real files; powerful, needs your oversight.

The expert in the building is the same regardless of which door you use. **Pick based on what you need to do** — not what sounds most technical.

 **This week's action:** open Claude.ai and use it for three real tasks you'd normally do yourself. Notice where it shines, where it needs guidance, where it gets things wrong.

[Read Theory →](#)

[Go to Homework →](#)
