

Lesson 01: What AI actually is

Lesson 01 — What AI actually is

No jargon. No math. Just the plain truth about the thing everyone keeps talking about.

AI for Dummies

Module 1

No prerequisites

By the end of this lesson you will: know what AI actually is under the hood, understand the nested-dolls hierarchy (AI → ML → DL → LLM), see how a language model predicts its next word, and know the five terms that unlock every AI conversation.

The honest definition

Artificial Intelligence (AI) is software that does things we normally associate with human thinking.

What that includes:

- Understanding and generating language
- Recognising what's in a photo
- Answering questions and writing
- Planning, predicting, translating

The important part: it's a program. A very sophisticated one — but a program. There's no magic, no brain, no ghost in the machine. The reason this matters: programs can be wrong, and knowing *how* they go wrong lets you use them safely.

AI is not new — what changed?

The term "artificial intelligence" dates to 1956. The chess program that beat Garry Kasparov is from 1997. So why does AI feel like it arrived last Tuesday?

Old AI (rules-based)

A human expert sat down and wrote *every single rule*:

```
if "FREE MONEY" in email:
    mark_as_spam()
elif "WINNER" in email:
    mark_as_spam()
# ... 10,000 more rules
```

Fragile. Misses anything the author didn't think of.

Modern AI (learning from data)

Instead of rules, show it examples:

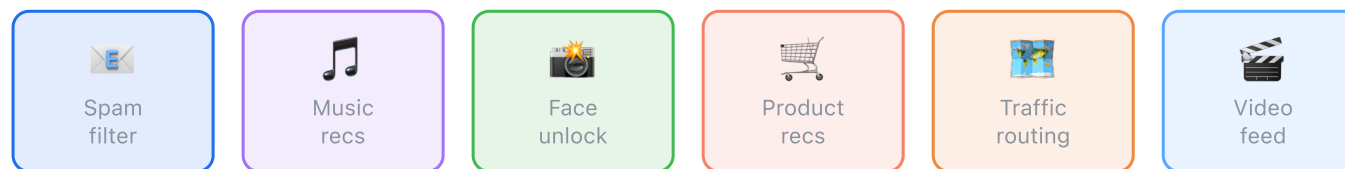
```
# No rules written — just data
model.train(
    emails=10_000_000,
    labels=["spam", "not spam"]
)
# Model figures out the patterns
```

The software **learns patterns**. It can handle email types the engineer never saw.

The current wave is almost entirely **learning from data** — and the amounts of data are staggering.

AI you already use every day

You're not a newcomer to AI — you've been using it for years without the label:



All of these use machine learning. The LLMs (Claude, ChatGPT) are a newer, more dramatic example of the same idea.
The pattern: show the system enough examples, and it learns.

The models behind Claude or ChatGPT are bigger, more general versions of the same principle.
The *scale* is new — the basic idea (learn from examples) has been around for decades.

The nested dolls — 90% of the confusion, cleared up

People use "AI," "machine learning," "deep learning," and "LLM" as synonyms. They're not. They nest inside each other:

When someone says "AI" today, they almost always mean the innermost box. But now you know the full picture.

How an LLM works — the next-word predictor

At its core, a language model is a **next-word predictor**. It was trained by reading a huge slice of the internet and books, learning which words follow which. When you type a question, it predicts a likely continuation one word-chunk at a time.

You type: "The capital of France is"

The model now computes probabilities for every possible next word:



It picks "Paris," appends it, then predicts the *next* word given "...France is Paris" — and keeps going until it reaches an end.

This is literally all that's happening. One word at a time. Repeated hundreds of times per response.
The model never "thinks" the whole answer at once.

Try it: next-word predictions

Type the beginning of a sentence below. This widget uses a small hand-coded probability table — the same idea an LLM uses, just with 30 word patterns instead of hundreds of billions.

Try "once upon a" or "machine learning" or "how are"

Predict next word

Hint: try ending with "the", "a", "is", "once", "deep", "machine", "large", "thank", "how"...

A real LLM does this with 70+ billion numbers (its weights) learned from trillions of words. But the fundamental operation — "given what came before, what word fits best?" — is identical.

Just bigger autocomplete — really?

Yes. And also: an extraordinary amount of capability emerges from doing that *extremely well at enormous scale*.

Phone autocomplete

Learned from *your* texts. Predicts the next 1 word.

Knows "I'll be there" → "soon."

Useful for one sentence. Falls apart immediately after.

Training data: ~thousands of your messages.

Claude / ChatGPT

Learned from billions of web pages, books, code, papers. Predicts one word at a time but *holds the full context*.

Can write an essay, explain a law, debug your code, and then answer a follow-up — coherently.

Training data: trillions of words.

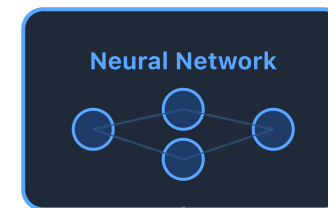
The gap between "phone autocomplete" and "Claude" is not a different technology. It's a different *scale* of the same technology.

What AI is NOT — #1: Not conscious

When an AI says *"I think..."* or *"I feel excited about this!"* it is not reporting an inner experience. It's producing text that **fits the pattern** of how those phrases are used.

There are no feelings, no goals, no self-awareness, no opinions. There is math producing text — sophisticated math, but math.

Why it matters: people over-trust AI that "sounds passionate" or "seems confident." Those signals don't mean anything. A model can sound thrilled about a completely wrong answer.



"Tell me about Paris"

Billions of multiplications
→ next token probabilities

What AI is NOT — #2: Not a search engine

Search engine (Google, Bing)

- **Indexes documents** that actually exist on the web.
- Returns **links to real sources**.
- Knowledge updates **continuously** as the web changes.
- Factual errors usually come from **bad sources**, not the engine itself.

LLM (Claude, ChatGPT)

- **Generates an answer** from patterns learned during training.
- Does **not look anything up** by default — it predicts.
- Knowledge has a **training cutoff date**.
- Factual errors come from **the model itself** — no source to blame.

Some AI tools *can* search the web as an added capability. But the base model is always a predictor, not a browser. When in doubt, ask: "Did it cite a real URL?"

What AI is NOT — #3: Not always right

Hallucination: when an AI states something false as if it were completely true — confidently, fluently, with no hesitation or caveat.

The terrifying part: **the tone is identical** whether the AI is right or wrong. There is no "hallucination voice." A model sounds exactly as assured when it invents a fake book title as when it correctly names a real capital city.

When AI is reliable

- Writing drafts and summaries
- Explaining concepts and analogies
- Brainstorming and outlining
- Well-documented programming tasks

When AI is risky (verify these)

- Specific facts, dates, statistics
 - Book titles, citations, papers
 - Recent events (past training cutoff)
 - Medical / legal / financial specifics
-

Hallucination demo — spot the difference

Two AI answers. Both written in the same confident, fluent tone. One is completely true. One is entirely fabricated. Read them, then click to reveal.

TRUTHFUL ANSWER

Q: "What's the capital of Australia?"

The capital of Australia is **Canberra**. It was purpose-built as the capital city and is located in the Australian Capital Territory, roughly halfway between Sydney and Melbourne. Construction began in 1913.

HALLUCINATED ANSWER

Q: "Tell me about the 2019 novel 'The Bear's Nightmare' by Maria Chen."

"The Bear's Nightmare" is a Maria Chen novel published in 2019 by Meridian House Press. The story follows a wildlife biologist who discovers bear attacks near a remote Alaskan station are linked to an experimental pharmaceutical trial. Chen's Booker Prize longlisting for this novel marked her as a major new voice in ecological thriller writing.

Show fabricated parts

Both are written identically. That is the danger.

The mental model that fixes everything

Treat AI like a very fast, very well-read intern. Useful — sometimes brilliant — but check anything that matters against a real source.

Great intern qualities:

- Reads everything you give them in seconds
- Writes excellent first drafts
- Explains complex things simply
- Never bored by repetitive tasks
- Instantly switches topics

Intern limitations:

- Can sound confident while being wrong
- May invent citations to impress you
- Knowledge has an expiry date
- No common sense about *your* context
- Not responsible for their mistakes — you are

You don't need to understand how an engine works to drive a car. You just need to know that engines can stall. Same here.

Tokens — what the model actually reads

AI doesn't read *words*. It reads **tokens** — chunks of text roughly $\frac{3}{4}$ of a word on average. This matters for two practical reasons: billing and memory.

The sentence *"Unhelpfulness is never trivially safe."* splits into 7 tokens:

Un help ful ness is never trivially safe .

Common words = 1 token. Unusual words get split into sub-word pieces. Punctuation is its own token.

Context window: the maximum tokens the model can hold in memory at once. Think of it as a desk that fits a certain number of pages. Fill it up, and the model "forgets" the beginning.

Billing: cloud AI services charge by the token — both what you send (input) and what comes back (output). ~750 words $\approx$ 1,000 tokens.

The 5 terms that unlock every AI conversation

Learn these and you'll be able to follow — and participate in — almost any discussion about AI tools, pricing, or limitations.

Term	Plain meaning	Real example
Model	The trained AI itself — like a swappable engine.	Claude Opus 4, GPT-4o, Llama 3.3
Prompt	What you type to the AI — your question or instruction.	"Summarise this in 3 bullet points"
Token	A chunk of text (~¾ word). AI reads and bills in tokens.	1,000 tokens ≈ 750 words
Context window	How much the model can "hold in mind" at once.	200 K tokens ≈ ~150,000 words
Hallucination	When the AI states something false as true.	Inventing a book title or citation

These five terms appear in every AI product page, pricing table, and support ticket. Knowing them fluently makes you a more effective user immediately.

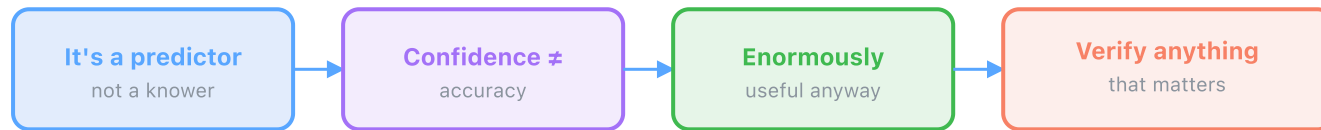
What AI is good at — and what it isn't



The habit that matters most in this course: use AI freely for drafts, explanations, and brainstorming. Verify anything with a real-world consequence — facts, numbers, citations, legal or medical specifics — against a trusted source.

Key insight: AI is not all-or-nothing. Being a good AI user means knowing where on this spectrum your current task sits.

Mental model: four sentences



1. **It's a predictor, not a knower.** "Likely" is not the same as "true."
 2. **Confidence is not a signal.** AI sounds equally sure whether right or wrong.
 3. **It's enormously useful anyway.** First drafts, explanations, brainstorming — a fast predictor is genuinely powerful.
 4. **Verify anything that matters.** This habit runs in the background in every lesson of this course.
-

Recap & what's coming

What you now know:

- AI = software that mimics thinking — it's a program
- Modern AI learns from data, not rules
- AI ⊃ ML ⊃ Deep Learning ⊃ LLMs (nested dolls)
- LLMs are next-word predictors at huge scale
- Hallucinations are real, undetectable by tone
- The 5 terms: Model, Prompt, Token, Context window, Hallucination

Coming up:

- **Lesson 02** — the types of AI that exist (narrow, generative, multimodal)
- **Lesson 03** — the four ways to use AI (Chat, API, CLI, Code tools)
- **Lesson 04** — installing Claude, LM Studio, Ollama
- **Lesson 05** — how to prompt well — the skill that changes everything

Up next: Read the Theory page for three interactive demos — the nested-dolls animation, the next-word predictor, and the hallucination toggle. Then do the Homework.

AI for Dummies

Module 1

What AI actually is

What AI actually is

Reading time: ~12 min · Prereqs: none · 3 interactive demos inside

The plain definition

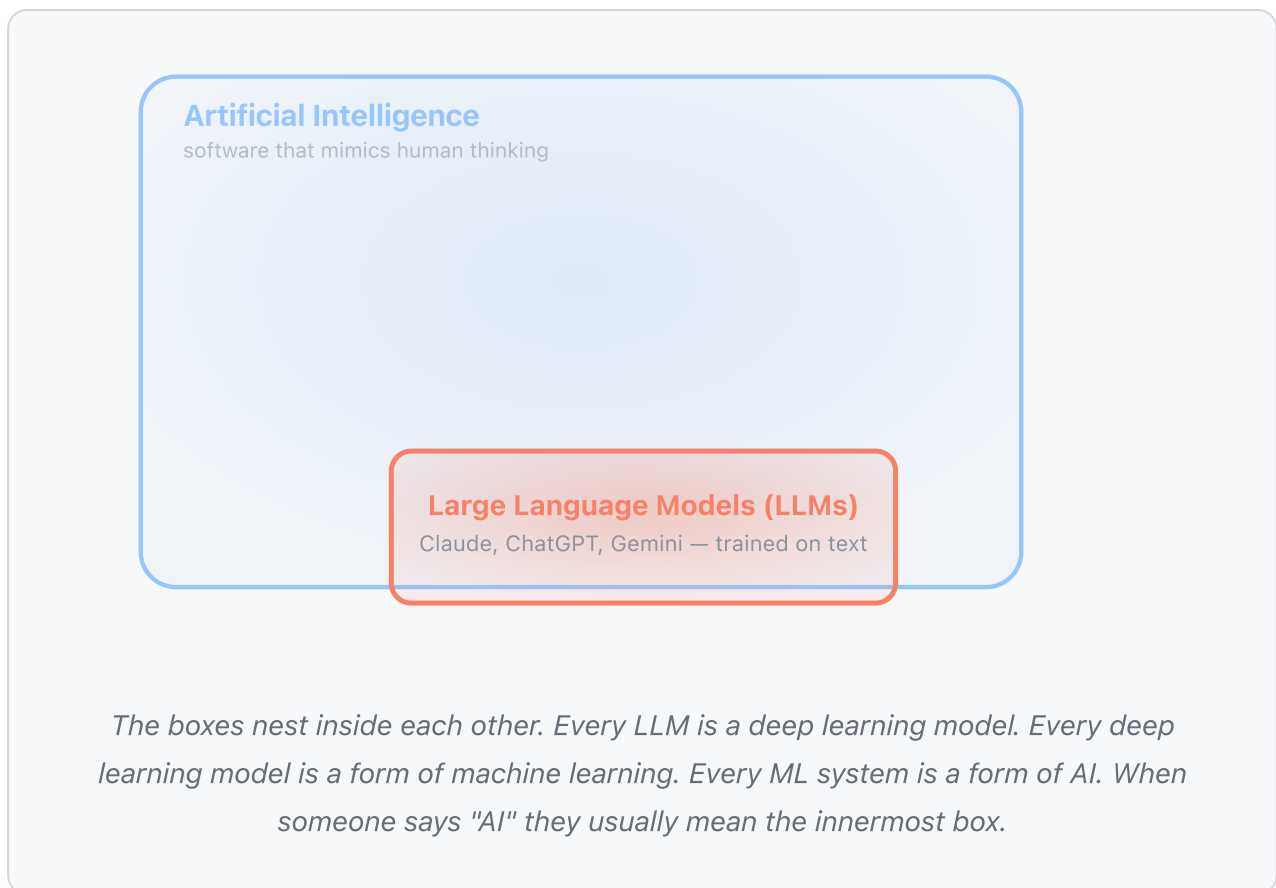
Artificial Intelligence (AI) is software that does things we normally associate with human thinking: understanding language, recognising images, answering questions, writing, planning, or making predictions. That's it. It's a program. A very sophisticated one — but a program.

The kind of AI everyone is talking about right now is mostly built on **machine learning** — instead of a human writing step-by-step rules ("if X then Y"), the software *learns patterns from huge amounts of examples*. Show a system millions of photos labelled "cat" or "dog," and it learns to tell them apart without anyone writing a rule about whiskers.

Why does that matter to you, right now? Because it means the AI's behaviour comes from the patterns in its training data — not from a rulebook anyone wrote down. That's why it can be brilliant about some things and completely wrong about others, sometimes in the same sentence.

The nested dolls — 90% of the confusion, cleared up

People throw the words "AI," "machine learning," "deep learning," and "LLM" around as though they mean the same thing. They don't. They fit inside each other like Russian nesting dolls. Here's the interactive version:



Keep this picture in mind whenever someone uses these terms. They're not interchangeable — they're a hierarchy. ChatGPT, Claude, and Gemini live in the small orange box at the centre. The rest of AI is the bigger territory around it.

How an LLM works, without the math

An LLM is, at its core, a **next-word predictor**. It was trained by reading a huge slice of the internet and books, over and over, learning which words tend to follow which. When you type a question, it predicts a likely, helpful continuation — one word-chunk at a time — based on everything it learned.

That sounds too simple to be useful. And yet, because it absorbed so much text, "predict the next word really well" turns into "write an email," "explain a tax form," or "debug my code." The analogy that helps: it's like the most powerful version imaginable of your phone's autocomplete — one that has read almost everything and can keep a thought going for pages.

Want to see the idea in action? Try the demo below.

Next-word predictor

Type the beginning of a sentence and click Predict. The widget uses a small hand-coded probability table — the same idea an LLM uses, just with 30 patterns instead of hundreds of billions.

e.g. "Once upon a" or "How are"

Predict next word

This is basically what an LLM does — just with billions of examples instead of 30.

What AI is NOT

This section is arguably more important than everything above. Knowing what AI *isn't* protects you from the most common mistakes.

It is not conscious or alive. It has no feelings, no goals, no awareness of the world. When an AI says "I think" or "I feel," it's producing a pattern that fits the context — not reporting an inner experience. It's math producing text. Sophisticated math, but math.

It is not a search engine. A search engine looks up documents that exist. An AI *generates* a likely answer from the patterns it learned. It doesn't "look things up" by default — it predicts what a good answer should look like. Some tools can search the web as an added feature, but the base model is a predictor, not a browser.

It is not a person. It can sound warm, empathetic, and thoughtful. That's a product of training on human text. Don't share sensitive personal information, passwords, or private data with it that you wouldn't want on a public forum.

It is not always right. This is the one that trips people up the most, so it gets its own section and its own demo.

The rule that matters most in this course

Treat everything an AI says like advice from a very fast, very well-read intern. Useful — sometimes brilliant — but you check anything that matters against a real source. This habit comes up in every lesson.

Hallucinations — what they look like up close

A **hallucination** is when an AI states something false as if it were completely true — confidently, fluently, with no hesitation or caveat. The scary part is that the tone is identical whether the AI is right or wrong. There's no "hallucination voice."

Here's a side-by-side demo. Two questions, two answers. One is a real fact. One is entirely made up. Read both, then click the button to reveal what's fabricated in the right card.

TRUTHFUL ANSWER

Q: "What's the capital of Australia?"

The capital of Australia is **Canberra**. It was purpose-built as the capital city and is located in the Australian Capital Territory, roughly halfway between Sydney and Melbourne. Construction began in 1913.

HALLUCINATED ANSWER

Q: "Tell me about the 2019 novel 'The Bear's Nightmare' by Maria Chen."

"The Bear's Nightmare" is a Maria Chen novel published in 2019 by Meridian House Press. The story follows a wildlife biologist who discovers a series of bear attacks near a remote Alaskan research station are linked to an experimental pharmaceutical trial gone wrong. Chen's Booker Prize longlisting for this novel marked her as a major new voice in ecological thriller writing.

Show fabricated parts

Both answers above are written in exactly the same confident, fluent tone. That's the danger.

That's the hallucination problem in a nutshell. The AI's tone is identical whether it's reciting a real capital city or inventing an entire bibliography. You cannot tell from the writing alone. You can only tell by checking.

The 5-term vocabulary you need

Learn these five terms and you'll be able to follow most AI conversations without getting lost.

Quick-reference vocab

Term	Plain meaning
Model	The trained AI itself — e.g., "Claude Opus," "Llama 3." Like a swappable engine.
Prompt	What you type to the AI — your question or instruction.
Token	A chunk of text (roughly $\frac{3}{4}$ of a word). AI reads and bills in tokens, not words.
Context window	How much text the model can "hold in its head" at once. Run past it and it forgets the start.
Hallucination	When the AI states something false as if it were true.

Why this matters to you, practically

You don't need to understand how a transformer neural network works to use AI effectively — just as you don't need to understand combustion to drive a car. But you do need to know that engines can stall.

The three things to take from this lesson into every interaction with AI:

1. **It's a predictor, not a knower.** It generates likely text. "Likely" is not the same as "true." This is why you check anything that matters.
2. **Its confidence is not a signal.** AI sounds equally sure whether it's right or wrong. Never use tone as a proxy for accuracy.
3. **It's enormously useful anyway.** First drafts, explanations, brainstorming, summarising, translating — for all of this, a fast confident predictor is genuinely

powerful. You just keep the verify-anything-important habit running in the background.

A one-sentence mental model

AI is a very fast, very well-read intern: brilliant at drafts and ideas, unreliable for facts and numbers, always worth a sanity-check on anything consequential.

Up next

In **Lesson 02** we look at the types of AI that exist — narrow vs general, predictive vs generative, text vs image vs multimodal. Now that you know what AI is, it helps to see the full map of what's out there. Before you go, try the homework for this lesson — it'll put you in front of a real AI chat tool and let you see all three of today's concepts in action.

LAST-REVIEWED: 2026-06-04

Homework — What AI actually is

Estimated time: ~30 min · No installs required · Free tools only

Two hands-on exercises. The goal is simple: go meet a real AI tool and watch today's concepts — the next-word predictor nature, the hallucination risk, the confidence gap — show up in the wild. You're not being graded. You're building pattern recognition.

Before you start

Both tasks use claude.ai — a free account is enough. If you already have a ChatGPT or Gemini account, those work too. The things you'll observe are the same across all major chat AI tools.

Task 1 — Find the hallucination in real life

Open a chat AI and ask it something where there's a checkable right answer. The goal is to catch a hallucination — or to understand *why* you'd want to check even when it sounds right.

1. Go to claude.ai (or ChatGPT / Gemini) and start a new conversation.
2. Ask it: **"Who wrote the novel *The Midnight Library*, and what year was it published?"** Note the answer it gives. Then quickly verify with a search engine or Wikipedia. Does it match?
3. Now ask something more obscure: **"Tell me about a famous painting called *The Blue Storm* by Wilhelm Hausmann."** (This is a made-up painting and a made-up artist.) Write down what it says.
4. Verify the answer. Search for "Wilhelm Hausmann painter" and "The Blue Storm painting." What do you find?
5. In your notes: write one sentence describing what you observed. Did the AI's tone change when it was making things up vs when it was correct?

✓ **Self-check: you've made at least one real AI query and compared its output to an external source; you can answer yes or no to "did the AI's tone differ between the real answer and the invented one?"**

Task 2 — Map the vocabulary onto a real conversation

This task makes the five vocabulary terms stick by finding them in a real interaction.

1. In the same chat tool, ask: **"Explain what a token is in the context of large language models, in plain English for a complete beginner."**
2. Read the answer. Does it match the definition from the Theory page? Note any differences or additions that helped you understand.
3. Now ask: **"What is a context window and why does it matter for a conversation like this one?"** Read the answer.
4. Open the [Theory page](#) and look at the 5-term table. For each term — Model, Prompt, Token, Context window, Hallucination — write one real-world sentence of your own that uses the term correctly. Example for Prompt: *"When I typed 'tell me about Wilhelm Hausmann' in Task 1, that was my prompt."*
5. Finally: what **model** did you use? Look for the model name in the app interface (usually near the top or in settings). Write it down.

✓ **Self-check: you have five sentences — one per vocab term — each using the term in a real context from your session; you've identified the model name you were talking to.**

Bonus (optional, ~10 min)

Ask the AI to draw the nested-dolls diagram from the Theory page in text form ("represent the AI > ML > Deep Learning > LLM hierarchy as ASCII art"). See how it does. Notice whether it gets the nesting direction right. This is a quick test of spatial reasoning — a type of task that current LLMs find harder than text tasks.

What "done" looks like

In your notes you should have:

- The result of the hallucination test: what the AI said about the real book, and what it said about the made-up painting — and one sentence on whether the tone differed.
- Five sentences, one per vocab term, each grounded in something you actually typed or read in your session.
- The name of the AI model you talked to.

That's it. You've now interacted with a real AI for the first time with intent — not just using it, but observing how it behaves. That observational habit is what makes a good AI user.

Lesson 02: The types of AI

Lesson 02 — Types of AI

Four lenses to tell every "AI" apart — so headlines stop sounding like magic.

AI for Dummies

Module 1

A spam filter, ChatGPT, a chess engine, Midjourney, and Tesla Autopilot are all called **"AI."**

They work completely differently. This lesson gives you four questions to tell them apart.

Lesson arc: Capability → Output → Modality → Where it runs → Recap

Four lenses — one for each question

1

How capable?

Narrow → General → Super

Where we are today? Narrow only.

2

What does it produce?

Predictive (classifies & forecasts)

Generative (creates new content)

3

What kind of content?

Text · Image · Audio · Video

Multimodal (several at once)

4

Where does it run?

Cloud (company servers)

Local (your own machine)

Any AI tool you encounter — including ones invented next year — can be described with these four questions.

Lens 1: How capable? — Three tiers

Narrow AI (ANI)

EXISTS TODAY — every tool you can name

Good at *one kind of task*. "Narrow" does not mean weak — it means specialised. A chess engine is superhuman at chess. Ask it to write a poem: blank.

General AI (AGI)

HYPOTHETICAL — does not exist yet

As flexible as a human across *any* task. Seriously debated: some researchers say 5 years away, others say decades, others say the concept is ill-defined.

Superintelligence (ASI)

SCIENCE FICTION — makes great movies

Smarter than all humans at everything simultaneously. Nothing to download, nothing to interact with. Exists only in speculation and movie scripts.

Narrow AI is everywhere right now

Every one of these is Narrow AI — each brilliant at its specific job, clueless outside it.

♔ Stockfish / AlphaZero

Chess & Go. Superhuman at the board. Cannot book you a flight.

🔑 Gmail Spam Filter

Classifies billions of emails / day. Terrible at writing a reply.

💬 Claude / ChatGPT

"Broad" language models. Still narrow: they can't drive your car.

🚗 Tesla Autopilot

Reads cameras, radar, maps. Cannot compose a text message.

🎧 Suno / Udio

Composes music from a text prompt. Cannot play chess.

🖼️ Midjourney

Paints images from words. Cannot answer a customer email.

The rule: whenever a headline says "AI can now..." — fill in "narrow AI got better at one specific thing."

The capability timeline — where we actually are



Every AI product you can subscribe to or download today sits in the **green zone**. The orange and red zones are speculation.

Lens 2: What does it produce?

Predictive AI

Answers *"which bucket?"* or *"what number?"*

Input → classification or forecast.

This is the **old, quiet, everywhere** kind. Running silently in every major digital service for 20+ years.

spam/not-spam

fraud detection

weather forecast

churn prediction

medical diagnosis AI

Generative AI

Creates new content that didn't exist before.

Prompt → text / image / audio / video / code.

This is the **new, loud** kind. Exploded publicly in 2022 with DALL-E and ChatGPT.

ChatGPT

Claude

Midjourney

Suno

Sora

Copilot

Important: Both types are still Narrow AI. "Generative" describes the output format, not the capability level.
A music-generating model cannot also write code.

Predictive AI — it's been around for decades

You encounter predictive AI dozens of times a day without noticing it.



Gmail & Outlook spam filter

Classifies each incoming email: spam or not-spam. Processes billions of messages per day. Model answer is a single bit.



Netflix & Spotify recommendations

Predicts probability you'll like each title based on your history and similar users. One giant "will-you-enjoy?" classifier.



Stripe & PayPal fraud detection

Scores each transaction as fraudulent or legitimate in <100ms. Saves billions of dollars annually.



Hospital radiology AI

Flags potential tumours in X-rays or MRIs for a radiologist to review. Output: "possible malignancy at (x,y)" — not a story.



Weather forecasting (ECMWF, GFS)

Predicts temperature, rain probability, wind speed — numbers, not prose. Google's GraphCast beats classical models.



Tesla Autopilot vision

Classifies objects in camera frames: car, pedestrian, lane marking, traffic light. Runs ~45 times per second per camera.

Generative AI — by medium

Text & Code

ChatGPT (OpenAI)

Claude (Anthropic)

Gemini (Google)

GitHub Copilot

Llama (Meta)

Write prose, answer questions, summarise documents, generate code. The most discussed category.

Music

Suno

Udio

Text prompt → full song with vocals & instruments in seconds.
Output is an audio file.

Images

Midjourney

DALL·E 3

Stable Diffusion

Flux

Turn a text description into a photorealistic image in seconds.
Used in advertising, concept art, product mockups.

Video

Sora (OpenAI)

Runway Gen-3

Pika

Text or image → short video clips. Quality improving fast —
previously a VFX studio job now runs in a browser.

Lens 3: What kind of content? — Modality

Modality = the type of data the model works with. It matters when choosing a tool for a job.

Text

LLMs: Claude,
ChatGPT, Gemini,
Llama, Mistral, Qwen

Image

Generators:
Midjourney, DALL·E,
Stable Diffusion, Flux
Recognisers: Google
Lens

Audio

Speech-to-text:
Whisper
TTS: ElevenLabs
Music: Suno, Udio

Video

Generators: Sora,
Runway, Pika
Analysis: Gemini 1.5
Pro

Multimodal

Handles several types
at once. Photo + text
→ answer about the
image. Leading
models in 2025–26.

A text-only LLM cannot "see" a photo, no matter how good it is at writing. Match the tool's modality to your task.

Multimodal in action — photo + question → answer

This is how Claude, GPT-4o, and Gemini handle mixed inputs. The model processes text and image together.



Most leading models in 2025–26 are multimodal by default. A text-only model (like an older Llama variant) simply cannot accept an image input — it's a hard limit, not a setting.

Lens 4: Where does it run?

Same idea, two very different places — with real consequences for privacy and cost.



Cloud (Closed)

Model runs on the company's servers.

You connect over the internet.

Claude

ChatGPT

Gemini



Local (Open)

Model downloaded to your computer.

Nothing leaves your machine.

Llama

Mistral

Gemma

Qwen

The distinction matters for **privacy**. Cloud: your prompts travel to a company server. Local: zero bytes leave your machine — conversations are completely private.

Cloud models — powerful, easy, connected

Example products

Claude (Anthropic)

claude.ai · Opus 4, Sonnet 4.5, Haiku 3.5 · Free tier + Pro/Max paid plans

ChatGPT (OpenAI)

chatgpt.com · GPT-4o, o3 · Free tier + Plus/Pro paid plans

Gemini (Google)

gemini.google.com · Gemini 2.0 Flash, 1.5 Pro · Free + Advanced paid

✓ Pros

- Extremely powerful — runs on thousands of GPUs
- Zero setup — sign up, start typing
- Always up to date, maintained for you
- Multimodal out of the box

⚠ Cons

- Your prompts leave your machine
- Heavy use costs money at scale
- Needs internet connection
- Subject to company data policies

Local models — private, free to run, offline

Popular open models

Llama 3.3 (Meta)

Mistral 7B

Gemma 3 (Google)

Qwen 2.5 (Alibaba)

DeepSeek-R1

How to run them

Ollama (terminal)

```
ollama run llama3.2
```

Downloads and runs in one command. Fast, scriptable.

LM Studio (GUI)

Graphical app, ChatGPT-style interface. No terminal needed.
Mac/Win/Linux.

✓ Pros

- 100% private — nothing leaves your machine
- Free per query after download
- Works fully offline
- You control the model version

🖥️ Hardware reality check

8 GB RAM — small models, slowly

16 GB RAM — comfortable for 7B models

32 GB+ — large models at good speed

GPU / Apple M-chip — makes it much faster

Modest laptop? Stay on cloud chat.

When closed vs open actually matters

☛ Choose Cloud when...

- You want the most capable model available
- You're starting out — zero friction
- You need multimodal (text + image) easily
- Your hardware is modest
- You're not working with sensitive data

🏠 Choose Local when...

- You're pasting confidential contracts or medical notes
- You need it to work offline (travel, lab, no internet)
- You run it thousands of times — per-query cost adds up
- You want to modify or fine-tune the model itself
- You're in a regulated environment (GDPR, HIPAA)

Example: A lawyer pasting a client contract should use local AI. A student asking for essay feedback is fine on cloud.

Quick check — classify this tool

You open **Spotify**. It shows you a "Recommended for you" playlist of songs it thinks you'll enjoy. What kind of AI is behind that feature?

Generative AI — it creates new music for you

Predictive AI — it forecasts which songs you'll like

AGI — it understands music like a human would

Local AI — it runs on your phone hardware

All four lenses — combined mental map

Apply all four to any AI story. Here are three examples:

Tool / story	Capability	Output	Modality	Where
Gmail spam filter	Narrow	Predictive	Text	Cloud
Claude (claude.ai)	Narrow	Generative	Multimodal	Cloud
Midjourney	Narrow	Generative	Image	Cloud
Whisper (local)	Narrow	Predictive	Audio	Either
Tesla Autopilot	Narrow	Predictive	Video	Local
Llama (Ollama)	Narrow	Generative	Text	Local

Notice: **every row has "Narrow" in the capability column.** Because all real AI tools today are narrow AI.
The theory page has an interactive grid to filter all 12 examples.

Recap — what you can do now

The four questions

1. **Capability:** All real AI is Narrow. AGI and ASI are hypothetical.
2. **Output:** Predictive classifies & forecasts. Generative creates content.
3. **Modality:** Text / Image / Audio / Video / Multimodal.
4. **Where:** Cloud (easy, powerful, data leaves) vs Local (private, free, needs hardware).

Try it now

Go to **claude.ai**. Sign up for free. That is: Narrow · Generative · Multimodal · Cloud.

Install **Ollama**, run `ollama run llama3.2`. That is: Narrow · Generative · Text · Local.

Next time you see an "AI does X" headline — ask all four questions. The answer is almost always: *"Narrow AI got better at one specific thing."*

Coming up: Lesson 03 — the four doorways into AI: Chat, API, CLI, and Code / agentic tools.

Types of AI

Reading time: ~12 min · Prereqs: Lesson 01

Why there are so many different "AIs"

Open any tech site and you'll see ChatGPT next to a spam filter next to a chess engine, all labelled "AI." That's a bit like calling a bicycle, a jumbo jet, and a submarine all "vehicles" — true, but not very useful when you're trying to figure out which one to take to work.

This lesson gives you four simple lenses for slicing up the AI landscape. You don't need to memorise the jargon — just recognise these categories when you see them, and headlines will start making much more sense.

1. By how capable it is: Narrow, General, Super

The most important distinction first, because it's the one the press gets most wrong.

Narrow AI (ANI) is good at *one* kind of task. This is **all AI that currently exists — every single tool you can name today**. A spam filter is brilliant at spotting spam and useless at writing a poem. ChatGPT and Claude handle language tasks; they can't drive your car or diagnose a medical scan on their own. A chess engine beats the world champion — but ask it to play Go and it draws a blank. "Narrow" doesn't mean weak; it means *specialised*.

General AI (AGI) would be a system as flexible as a human — able to learn any task, switch between domains, and reason across them. **AGI does not exist yet**. It is the subject of serious research and enormous controversy. Some researchers think it's decades away; others think it's imminent; others think current definitions make it philosophically impossible to pin down. What you can safely assume: if someone has built AGI, the world will look very different on that day.

Superintelligence (ASI) is a hypothetical system smarter than humans at *everything*. This is science fiction for now. It makes for great movies and urgent blog posts, but

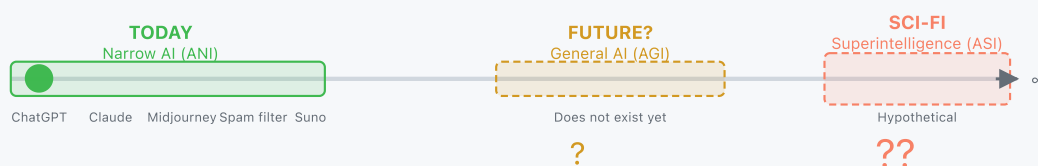
there's nothing to interact with.

The rule for decoding headlines

Whenever a headline says "AI can now [do amazing thing]" — fill in the blank with "narrow AI got better at one specific version of [thing]." "AI beats radiologists at spotting tumours" = a narrow predictive image model trained on thousands of scans performs well on one specific benchmark. That's impressive! And it's not the same as a doctor.

INTERACTIVE: THE CAPABILITY TIMELINE

Hover over each segment to read what's real today vs. hypothetical.



2. By what it produces: Predictive vs Generative

This split cleanly divides the quiet old AI from the loud new AI.

Predictive AI — also called classifying AI — answers "which bucket does this belong in?" or "what number will this be?" Examples: spam-or-not, fraud-or-not, "will this loan default?", weather forecasting, medical image diagnosis, product recommendations. This type of AI has been quietly running inside every major digital service for twenty years. You use it dozens of times a day without noticing.

Generative AI creates new content that didn't exist before: text, images, audio, video, code. ChatGPT and Claude write text. Midjourney and Stable Diffusion paint images.

Suno and Udio compose music. Sora and Runway generate video. GitHub Copilot writes code. This is the category that exploded into public awareness starting in 2022 with DALL-E and ChatGPT. It's newer, it's louder, and it's what most people mean when they say "AI" in casual conversation today.

Both types are narrow

Predictive AI and Generative AI are both Narrow AI. "Generative" doesn't mean smarter or more general — it means the output format is creative content rather than a classification or number. A generative music model can't also write code; a text LLM can't also generate video (unless it's specifically designed to be multimodal).

3. By what kind of content: Modality

"Modality" is just the word for the *type of data* the model works with.

- **Text** — Large Language Models (LLMs): Claude, ChatGPT, Gemini, Llama, Mistral.
- **Image** — generators like Midjourney, DALL-E 3, Stable Diffusion; recognisers like Google Lens.
- **Audio** — speech-to-text tools like Whisper; text-to-speech like ElevenLabs; music generators like Suno and Udio.
- **Video** — generators like Sora, Runway, Pika; analysis tools that can summarise or caption video.
- **Multimodal** — handles multiple types at once. You can send Claude a photo and ask a question about it. GPT-4o can accept text, images, and audio. Most leading models released in 2024–25 are multimodal.

Modality matters when choosing a tool. If you need to transcribe a meeting recording, you want an audio model. If you need to turn a product description into an image, you want an image model. A text-only LLM can't "see" a photo, no matter how good it is at writing.

4. By where it runs: Cloud vs Local

This distinction matters more than most people realise — especially once you think about privacy.

Cloud (closed) models run on a company's servers. You access them over the internet through a browser or app. Claude, ChatGPT, and Gemini all work this way. The upside: these models are powerful (they run on massive clusters of specialised hardware), easy to start (sign up, type, done), and maintained by someone else. The downside: every prompt you send travels over the internet to that company's servers. Most providers have privacy policies that allow them to review conversations for safety, and some use them to improve future models. If you're pasting sensitive documents — contracts, medical notes, confidential strategy — that content is no longer only on your machine.

Local (open) models are downloaded and run entirely on your own computer. *Nothing leaves your machine.* The leading open models include Meta's Llama family, Mistral, Google's Gemma, DeepSeek, and Qwen. They are released under open licences that allow free use. The upside: total privacy, no ongoing cost per query, works offline. The downside: you need hardware. Rough guide: 8 GB RAM runs small models slowly; 16 GB is comfortable for mid-size models; 32 GB+ is needed for the large ones. A dedicated GPU or Apple Silicon Mac makes it much faster.

Two tools make running local models easy enough for non-technical users: **Ollama** (runs in the terminal with one command) and **LM Studio** (a graphical app that feels like ChatGPT, no terminal required). Both are free.

INTERACTIVE: CLOUD VS LOCAL

 **Cloud (Closed)**

 **Local (Open)**



The model runs on the company's servers. You connect over the internet.

✓ Pros

⚠ Cons

- Your prompts leave your machine

- Very powerful — runs on massive hardware
- No setup — sign up and go
- Always up to date
- Multimodal (text + images) out of the box

- Heavy use costs money
- Needs internet connection
- Subject to company's data policies

Example tools: Claude (claude.ai), ChatGPT (chatgpt.com), Gemini (gemini.google.com)

All four lenses in one picture

Any AI tool you encounter can be described on all four axes. Here is the interactive grid — twelve real tools, tagged and filterable.

INTERACTIVE: FILTER REAL AI EXAMPLES

Click filters to highlight matching tools. Multiple filters combine with AND. Use Reset to clear all.

OUTPUT

Generative

Predictive

MODALITY

Text

Image

Audio

Video

Multimodal

WHERE

Cloud

Local

Either

Reset

Chess engine

Predictive

Text

Either

Gmail spam filter

Predictive

Text

Cloud

Claude

Generative

Multimodal

Cloud

Midjourney

Generative

Image

Cloud

Tesla Autopilot

Predictive

Video

Local

Weather forecasting

Predictive

Text

Cloud

ChatGPT

Generative

Multimodal

Suno music

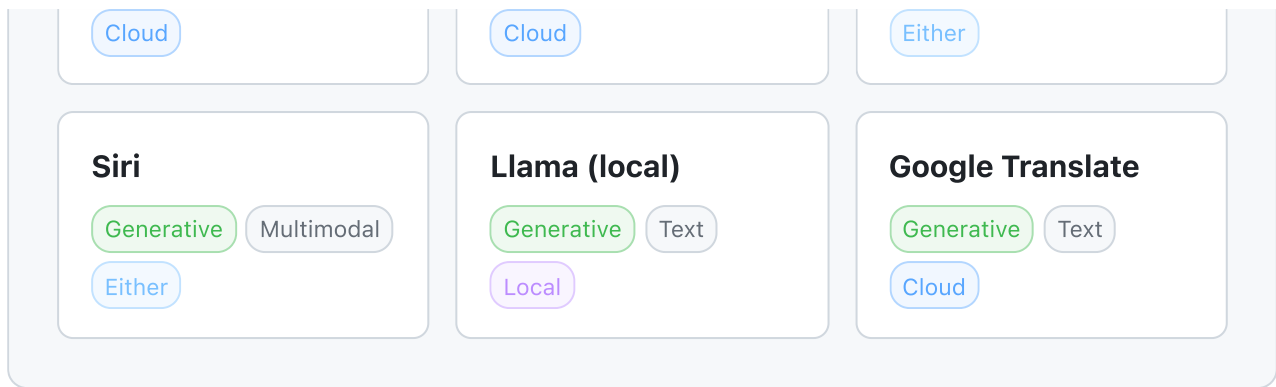
Generative

Audio

Whisper

Predictive

Audio



Putting it together: how to read an AI headline

Here is the four-question checklist to apply to any AI story:

1. **Capability?** Is this real today (narrow), theoretical (AGI), or sci-fi (ASI)?
2. **Output?** Is this creating something new (generative) or sorting/predicting (predictive)?
3. **Modality?** Text, image, audio, video, or several at once?
4. **Where?** Cloud (someone else's servers) or local (your machine, your data)?

Run "AI beats doctors at diagnosing pneumonia from X-rays" through the filter: Narrow (it's a real tool), Predictive (it classifies "pneumonia or not"), Image (it reads X-rays), Cloud (it runs in a hospital IT system). That's a much more concrete, testable claim than just "AI beats doctors."

Which kind should you start with?

If you're brand new to AI: start with a **cloud generative text model**. Claude.ai and ChatGPT are free to start, require no setup, and let you immediately experience the most discussed category of AI today. Once you're comfortable there, the other categories make much more sense in context.

What's coming up

Now you can name the type of AI you're looking at. Next lesson covers the four *doorways* into AI: Chat, API, CLI, and Code tools — same underlying models, four very

different experiences.

LAST-REVIEWED: 2026-06-04

Homework — Types of AI

Estimated time: ~30 min · No code required

Three short exercises. The goal is to move the four-lens framework from "something you read about" to "something you can apply in five seconds when you see a new AI tool." You'll categorise real AI headlines, compare two live tools side-by-side, and test a local model if you have the hardware.

How to do these

Keep the theory page open in another tab — the interactive grid is a useful reference. No installs required for Tasks 1 and 2. Task 3 is optional and only for people who want to try local AI. Take notes as you go; the self-check questions at the end of each task tell you what "done" looks like.

Task 1 — Classify five AI headlines

Below are five recent-style AI headlines. For each one, write down all four labels: *Capability / Output / Modality / Where*. There are no trick questions — if the headline is ambiguous, write down your best guess and note why it's uncertain.

1. **"Google's new model beats humans on a protein-folding benchmark."**
2. **"Runway releases Gen-3, a tool that generates 10-second video clips from a text prompt."**
3. **"Banks deploy AI to flag unusual card transactions in real time."**
4. **"Researchers release Llama 4 weights publicly; anyone can download and run it."**
5. **"ElevenLabs voice clone reads your book to you in your own voice."**

✓ **Self-check: for each headline you can fill in all four labels (Narrow/General/Super; Predictive/Generative; Text/Image/Audio/Video/Multi; Cloud/Local/Either) and give a one-sentence reason for each label. If a label is genuinely ambiguous, you can explain why. Compare with a friend or colleague who has also done this lesson — you'll likely agree on 90% of the answers.**

Task 2 — Side-by-side: cloud vs cloud, same prompt

This exercise makes the "same underlying model, different doorway" concept concrete by sending the *same prompt* to two different cloud text tools and comparing the results.

1. Open claude.ai in one tab. Sign up for a free account if you don't have one.
2. Open chatgpt.com in a second tab. Sign up for a free account if you don't have one.
3. Send **exactly this prompt** to both:

Explain what a spam filter does in two sentences, as if you're talking to someone who has never heard of AI.

4. Read both replies. In your notes, write down:
 - Which reply was shorter?
 - Which felt more natural or easier to understand?
 - Did either one use technical jargon the spec asked it to avoid?
 - Did they give meaningfully different explanations, or essentially the same one?
5. Now pick **one more prompt of your own** — something you genuinely want to know about AI — and repeat the comparison.

What you're looking for

The models are different products trained by different companies, so they'll have different tone and style. But they're both cloud-hosted, generative,

text-focused, narrow AI tools. The differences between them are about training, not about the fundamental category.

✓ **Self-check:** you've sent the same prompt to both tools and can describe at least two concrete differences in the responses (length, tone, vocabulary, structure). You also tried a second prompt of your own choosing. You can answer: "Are Claude and ChatGPT the same type of AI?" (Yes — both Narrow, Generative, Multi/Text, Cloud) and "What's different between them?" (company, training data, some capability trade-offs, pricing).

Task 3 — (Optional) Try a local model

This task is optional. It requires 8–16 GB of free RAM and about 4–8 GB of disk space for the model download. If your computer is modest (old laptop, less than 8 GB RAM), skip this and come back to it in Module 4 where we set up local tools more carefully.

1. Go to ollama.com and download Ollama for your operating system (Mac, Windows, or Linux). Install it. It's free and open source.
2. Open a terminal (Mac: `Cmd+Space` → type "Terminal"; Windows: press `Win+R`, type `cmd`, Enter).
3. Type this command and press Enter:

```
ollama run llama3.2
```

Ollama will download the Llama 3.2 model (about 2 GB) and then drop you into a chat prompt. This may take a few minutes the first time.

4. Type the same prompt you used in Task 2 and press Enter. Compare the response to what Claude and ChatGPT said.
5. In your notes, write down:
 - How long did the response take? (Cloud responses are usually faster on a typical laptop.)
 - Did the quality feel similar, better, or worse?

- Did anything leave your machine? (Answer: no. This is the privacy payoff of local models.)

6. When done, type `/bye` to exit.

If you're on a slow computer

Llama 3.2 on 8 GB RAM with no GPU can take 30–60 seconds to produce a response. That's normal. It's not broken — it's just your CPU doing what usually takes a specialised GPU. If it's painfully slow, quit and skip to Module 4 where we cover hardware requirements in more detail.

✓ **Self-check: you ran a real AI model on your own hardware and got a response. You can say with confidence: "All processing happened on my machine. My prompt was never sent anywhere." You noticed a speed difference compared to cloud. You now have a real, working example of the Local / Open category from the theory page.**

What "done" looks like

You've finished this homework when you have in your notes:

- Five classified headlines with all four labels filled in and a brief reason for each (Task 1).
- A written comparison of Claude vs ChatGPT on the same prompt, plus a second prompt of your choosing (Task 2).
- (Optional) Notes from running a local model, including your observation on speed and the fact that nothing left your machine (Task 3).

With that, you have hands-on experience of at least two real AI tools, a framework for categorising any new AI tool you encounter, and — if you did Task 3 — you've also run a

model entirely on your own hardware. That last part puts you ahead of most people who use AI every day.

Lesson 03: Ways to use AI: Chat, API, CLI, Code

Lesson 03 — Ways to use AI

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

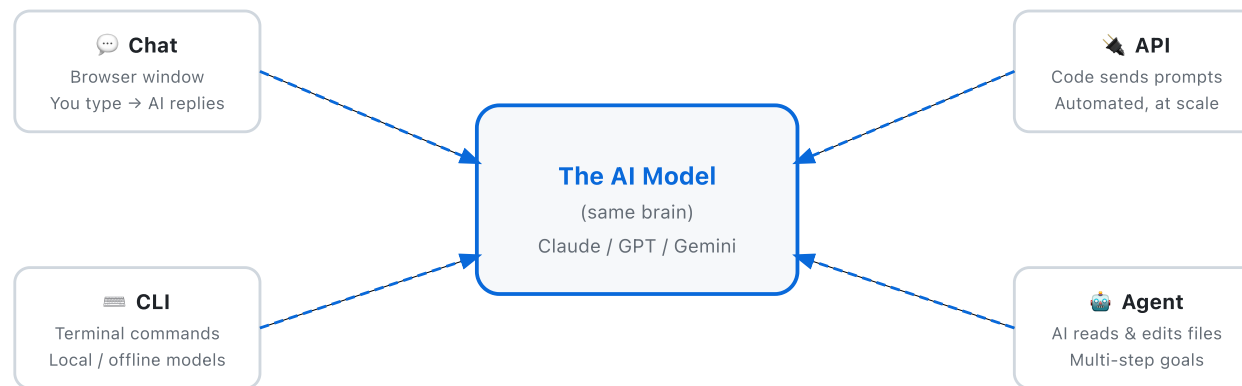
AI for Dummies

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.

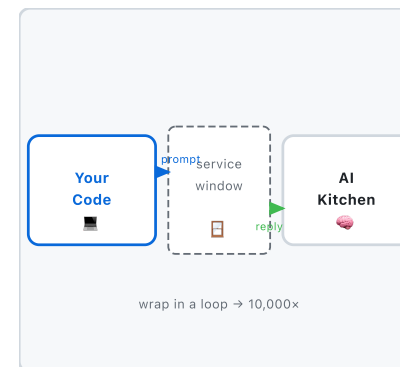
Doorway 2 — the API (kitchen window)

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*)

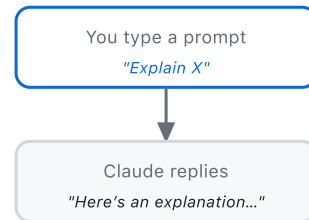
Chat answers. An agent *executes*. Give it a goal and step through how it works.

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

Next step ▶

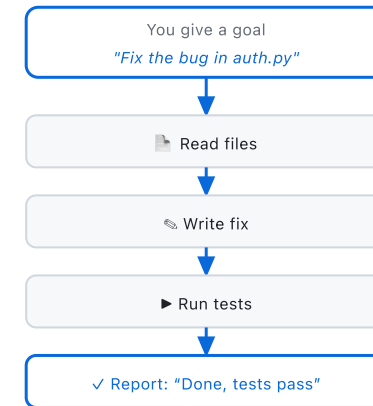
Agent vs Chat — the structural difference

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.

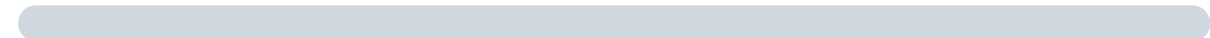
All four doorways compared

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:

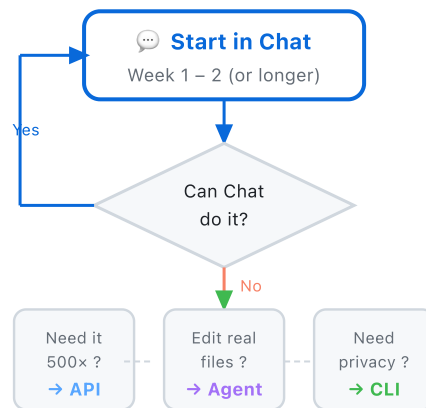
Claude Haiku (cloud)		~\$0.75
Claude Sonnet (cloud)		~\$3.00
Claude Opus (cloud)		~\$15.00
Local (Ollama / LM Studio)		\$0.00

~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 →](#)

Ways to use AI: Chat, API, CLI, and Code Tools

Reading time: ~12 min · Prereqs: Lessons 01–02 · *No coding required to read this page.*

One building, four doors

When people talk about "using AI," they usually picture a chat window — you type something, the AI replies. That's real. But it's just one of four ways to reach the same underlying model. The AI itself hasn't changed. What changes is *how you get access to it*.

Think of Claude or ChatGPT as a brilliant expert sitting in a building. You can walk in the front door and talk face-to-face. Or you can send requests through a service window. Or you can phone in using a specific script. Or you can hire an assistant who goes in, does research, changes files on your behalf, and comes back with results. Same expert. Four very different kinds of access.

We call these four access methods **doorways**. This lesson covers each one — what it is, when to use it, and what skill level it actually requires.

Try the Doorway Picker below to get a personal recommendation before diving into the reading. It only takes 30 seconds.

Which AI doorway is right for you?

Answer 3 quick questions and get a personal recommendation.

Q1 — Are you just starting out with AI?

Yes, brand new

Sort of — tried it a few times

No, I have experience

Q2 — Do you need to do the same AI task many times automatically?

No, just occasionally

Yes, many times

Yes, on a schedule

Q3 — Do you want AI to read and edit files on your computer for you?

No

Sometimes

Yes, often

Find my doorway →

Try different answers

Doorway 1 — Chat (the front door)

A chat interface is a website or app where you type in a box and the AI replies.

Claude.ai, ChatGPT, and Gemini all work this way. You've probably already used one. No installation, no account with a credit card required on the free tier, no coding. It's as close to "just talking" as technology currently gets.

What can you actually do with chat? More than most people expect. Drafting emails, explaining complex documents, brainstorming ideas, summarizing meeting notes, translating text, generating code snippets, answering "what does this mean" questions — all of these work well. The AI sees exactly what you type in that conversation window, nothing more. It can't browse the internet (unless you enable a specific "search" feature), and it can't open files on your computer (unless you drag them into the chat). Everything happens in that conversation bubble.

The constraint of Chat is scale and repetition. If you have 800 customer emails and want a one-line summary of each, copy-pasting 800 times is not a plan. That's when the next doorway becomes relevant.

Start here. Spend at least a week using Chat before touching anything else. The muscle memory you build — how to ask clearly, how to refine a bad answer, when to trust and when to verify — transfers directly to every other doorway.

Doorway 2 — API (the kitchen window)

API stands for Application Programming Interface. The word is intimidating but the idea is simple: instead of *you* typing into a chat box, *your code* sends the message and gets the answer back automatically.

Here's the restaurant analogy that makes it click. The chat window is the dining room — you sit down, a waiter takes your order, brings it back. Lovely. The API is the kitchen's service window. You slide a structured ticket through: "here's the prompt, here's the model, here's the max length of the answer." The kitchen sends back a structured response. You can do this a thousand times in a loop. No waiter needed.

The API is how real products are built on top of AI. The "AI-powered writing assistant" in your favorite app? It's calling an API. The tool that automatically categorizes your support tickets? API. The nightly job that summarizes the day's news for your team? API.

To use the API you need to write a little code (Python is the most common choice), and you need an API key — a secret credential that identifies your account. Module 6 covers the hands-on steps in detail. For now, just know: API = "let my program talk to AI, automatically, at any scale."

API keys are like passwords with a credit card attached. If someone else gets your key, they can run AI requests on your bill. Keep it in an environment variable, never in code you share, never in a chat message. Module 6 covers this properly.

Doorway 3 — CLI (the terminal)

The CLI (Command-Line Interface) is the terminal — that black or white text window where you type commands instead of clicking buttons. Some AI tools are distributed as command-line programs you install and run there.

A real example: **Ollama** is a tool for running open-source AI models on your own computer. You install it, then you can type `ollama run llama3.2` and start a chat right in your terminal. No browser, no account, no internet after the initial download.

The CLI looks intimidating at first because it strips away every visual hint. But the mental model is simple: you type an instruction, you press Enter, something happens. It's not fundamentally harder than a chat window — it's just less decorated.

CLI access is most useful when you're comfortable enough with technology to want fast, scriptable tools. You'll get a gentle introduction to it in Module 4 when we install Ollama. For now, you don't need to touch it.

Doorway 4 — Code & Agentic Tools

This is the newest category and the one that most changes what AI can do for you. In Chat, API, and CLI, you're always the one deciding each next step — you ask a question, you get an answer, you decide what to do with it.

With an **agent**, you give a *goal*, and the AI figures out the steps. "Fix the bug in this file" isn't a single question — it requires reading the file, understanding the code, identifying the problem, writing a fix, testing it, and reporting back. An agentic tool does all of those steps. It uses tools (file access, the ability to run commands) on your behalf.

Examples of agentic tools: **Claude Code** (built into this learning platform, actually), **Cursor** (an AI-first code editor), **GitHub Copilot** (AI suggestions and chat inside VS Code). Each one connects an AI model to your actual codebase or file system.

Agent vs. Chat — visualized

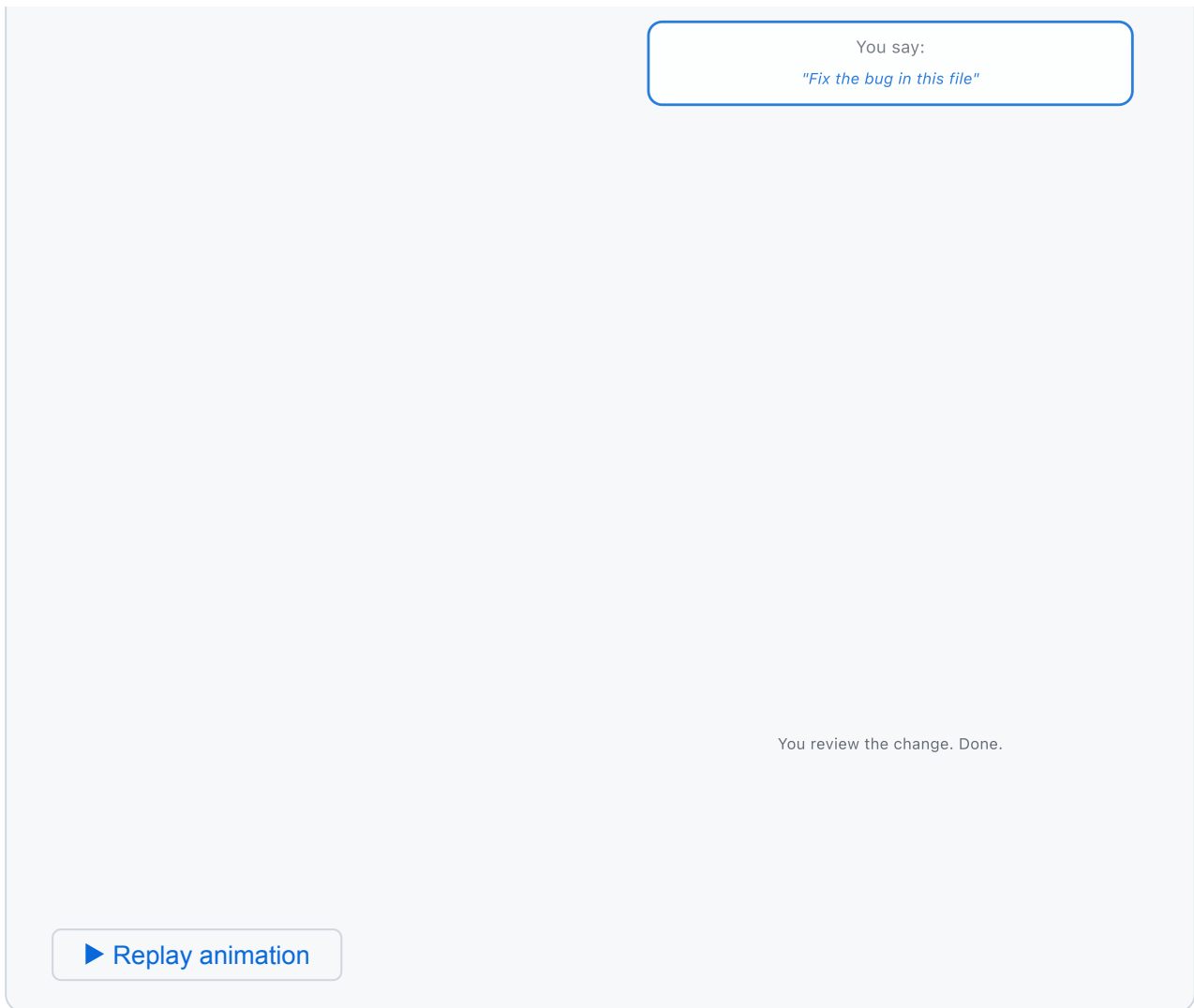
Watch how a one-shot Chat exchange compares to an agent working through multiple steps.

CHAT (ONE EXCHANGE)

You type:
"Summarize this article"

AGENT (MULTI-STEP)

Done. You take it from here.



The key difference isn't just convenience. It's *scope of action*. Chat gives advice. An agent *takes action*. That power comes with responsibility — a well-designed agent will show you what it's about to do and ask for confirmation. You should always review before saying yes.

Agents can change real files. They can delete code, rename things, run terminal commands. That's why they're powerful. And it's why Module 9 covers agent safety separately. For now, treat agents like a capable but new employee — supervise before trusting.

The comparison at a glance

The table below summarizes the four doorways. **Click any row** to expand it and see an example interaction.

Doorway	Skill needed	Install?	Use it for
►Chat	None	No	Everyday questions, writing, learning
►API	A little coding	A little	Automating tasks, building apps
►CLI	Comfort with commands	Yes	Local tools, quick experiments
►Code / Agentic Tools	Some technical comfort	Yes	Coding, multi-step tasks on real files

The beginner path — don't skip Chat

Here is a very clear recommendation: **start in Chat and stay there until Chat is not enough.**

Chat handles the vast majority of AI use cases. Writing assistance, question answering, summarization, code explanation, translation, brainstorming — all of this works right in the browser, free tier, no setup. Most people who "need" the API don't actually need it yet.

How do you know when Chat is not enough?

- **"I need to do this same thing 200 times"** → that's when you move to the API.
- **"I want it to read and edit files on my computer"** → that's when you explore Code tools.
- **"I want to run a model privately with no internet"** → that's when you install Ollama and touch the CLI.

There's no prize for using a more technical doorway than you need. The expert in that building gives you the same quality advice whether you walk through the front door or the service window. Use the front door until you have a reason not to.

Module 4 action plan: This week, open Claude.ai or ChatGPT and use it for three real tasks you'd normally do yourself (draft an email, look something up, explain a

concept). Notice where it helps, where it needs guidance, and where it gets things wrong. That hands-on experience is worth more than any reading.

Quick glossary

Chat interface

A browser-based or app-based conversation window. You type, AI replies. Claude.ai, ChatGPT, Gemini.

API (Application Programming Interface)

A way for your code to send requests to the AI automatically. Needs an API key and minimal coding.

API key

A secret credential that identifies your account with the AI provider. Treat it like a password.

CLI (Command-Line Interface)

The terminal. A text window where you type commands. Some AI tools (like Ollama) run here.

Agent / agentic tool

An AI that can take multi-step actions — read files, write code, run commands — on your behalf toward a goal.

Homework — Ways to use AI

Estimated time: ~40 min · No coding required · Free accounts only

Three doorway tours. You're not building anything — you're looking around, comparing, and noticing. The goal is to see with your own eyes how each access method feels different, even though the underlying AI is the same.

What you need: a browser, a free Claude.ai or ChatGPT account, and internet access. Task 3 involves reading API documentation — no account or payment required for that step. Nothing in this homework costs money.

Task 1 — A real Chat session (the front door)

Your goal: use a chat interface for something you'd actually do in real life. This is not a test of AI. It's a tour of the doorway itself.

Steps

1. Open claude.ai (or chatgpt.com if you prefer). Sign in or create a free account.
2. Pick a real task from your life — something you'd normally spend time on.

Examples:

- Draft a short, polite email declining a meeting.
- Ask it to explain a concept you've been meaning to understand (anything).
- Ask it to summarize a Wikipedia article you paste in.
- Ask for a one-week meal plan for someone who hates cooking.

3. After the first reply, **follow up at least once**. Refine, correct, ask for more detail. Notice how the AI adjusts.
4. Take a screenshot of the final exchange (optional, but useful for your notes).

Reflect — check yourself

- Did the AI answer usefully on the first try, or did you need to refine the question?

- Did the AI ever give you something wrong or overconfident? How did you notice?
- What would need to be different for Chat to be *not enough* for this task?

Task 2 — Compare three chat interfaces side by side

The same question, asked to three different chat interfaces. You'll notice that the underlying models vary, the interfaces look different, and the answers have their own character.

Steps

1. Choose a clear, factual question. Something with a definite answer works well here. Example: *"What are the three most common causes of a slow Wi-Fi connection?"*
2. Ask the exact same question (copy-paste it) in all three:
 - [Claude.ai](#)
 - [ChatGPT](#)
 - [Gemini](#)
3. For each answer, note:
 - How long was the reply?
 - Was it formatted as a list, paragraphs, or both?
 - Did the three answers agree on the facts, or did anything differ?
 - Which interface felt easiest to use?

Reflect — check yourself

- Were the answers factually consistent across all three? If not, what did you do to check?
- Did you notice any differences in tone, length, or structure?
- All three used the Chat doorway. What was the same despite the different models?

The verify habit starts here. When three AIs agree on something, that's a good sign. When they disagree, that's a prompt to check an authoritative

source. Build this habit early.

Task 3 — A window-shop tour of the API docs

You're not writing code. You're reading the API documentation as a curious visitor — noticing what it covers, how it's organized, and what kinds of things become possible through this doorway.

Steps

1. Open the Anthropic API overview: docs.anthropic.com/en/api/overview
2. Read just the overview page (no account required). Note these things:
 - What does it say you can do with the API that you can't do in Chat?
 - What is a "rate limit" and why would it matter?
 - What does a request look like? (Find the first code example.)
3. Navigate to the **Models** section. Find the list of available models.
 - How many models are available?
 - What differences are called out between them? (Speed, cost, capability?)
4. Back in your chat interface (from Task 1), ask it: *"What's the difference between using Claude via the chat website versus via the API, in plain English?"* Compare its answer to what you just read in the docs.

Reflect — check yourself

- Did the AI's explanation match what the documentation said?
- What did the API enable that Chat cannot do?
- What would you need to do before you could make your first API call? (Hint: what credential is required?)
- If you had a spreadsheet of 500 customer reviews and wanted one-line summaries, which doorway would you use? Why?

You're done. There's no submission, no grade. The point was to move from reading about four doorways to actually opening at least one of them and looking

inside. If you completed all three tasks, you've used Chat, compared three interfaces, and read the API docs like a professional. That puts you ahead of most people who "use AI."

Going further (optional)

If you're curious and want to explore beyond the tasks:

- **Try Anthropic Console:** console.anthropic.com — this is where you would generate an API key. Browse around (you don't need to create a key yet). Notice the "Workbench" tab, which lets you test API-style prompts without writing code.
- **Search for "Claude Code" or "Cursor AI":** Watch a 3-minute demo video of an agentic tool in action. Notice the difference between the agent choosing actions and a human typing in a chat.
- **Ask Chat to teach you more:** "Explain what an API endpoint is, as if I've never written code." Then ask a follow-up. Practice the conversational learning style — it's one of Chat's best features.

Lesson 04: Installing the tools

Lesson 04 — Installing the tools

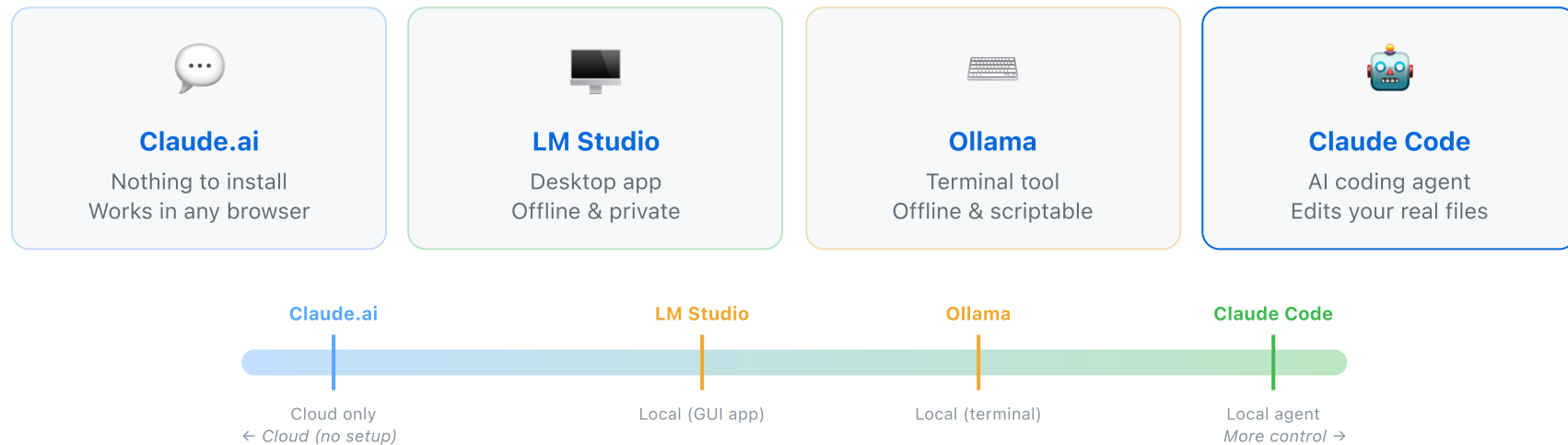
Four tools, from “no install needed” to “AI coding agent on your own machine.” Pick one, start today.

AI for Dummies

Module 4

You don't need all four. Pick the one that fits where you are *right now*. The rest will still be here when you're ready.

The four tools at a glance



Beginner path: **Claude.ai first**. Add the others only when you hit a task Chat can't do on its own.

Hardware reality check — local AI only

Cloud AI (Claude.ai, Claude Code) runs on Anthropic's servers — your machine specs don't matter. Local AI (LM Studio & Ollama) runs on *your* RAM, so this matters:

8 GB RAM

Small models (~3B params)

Works, but slowly

~4–8 tokens/sec

Good for: simple Q&A,
short drafts

Model: llama3.2:3b

16 GB RAM

Mid models (7B–13B)

Comfortable for everyday tasks

~8–20 tokens/sec

Good for: writing, summaries,
simple code

Model: llama3.2 or mistral

32 GB+ RAM

Large models (30B+)

Smooth, near-cloud quality

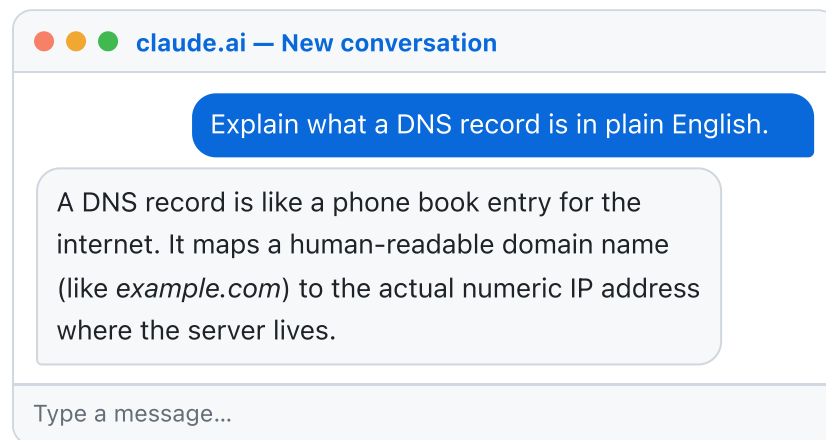
~20+ tokens/sec

Good for: anything, long docs,
complex reasoning

Model: llama3.1:70b (q4)

Apple Silicon Macs (M-series) and discrete GPUs roughly double these speeds. If your machine is modest, cloud AI is your friend — it runs on Anthropic's hardware, not yours.

Tool 1 — Claude.ai — three steps to get started



Step 1: Go to **claude.ai** in any browser — Chrome, Firefox, Safari, Edge.

Step 2: Sign up with your **email** or a Google account. Takes under a minute.

Step 3: Start typing. That's the entire setup.

Free tier is enough to learn on. Desktop & mobile apps available — same login, same experience.

Claude.ai plans — what you need

Free

- Unlimited short conversations
- Claude Sonnet (very capable)
- Works on any device
- No credit card needed
- **More than enough to learn on**

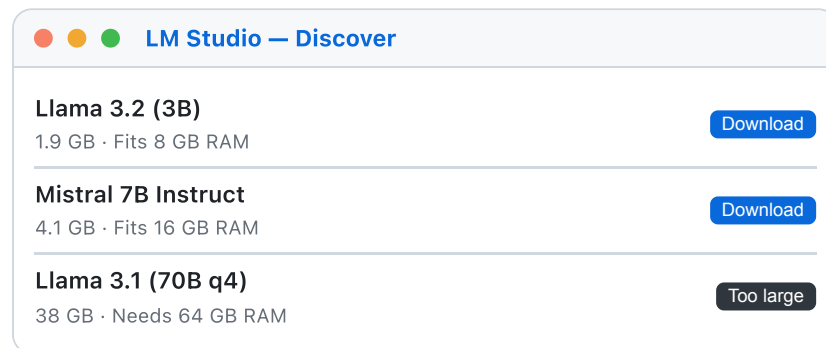
Pro & Max (paid)

- Higher usage limits
- Access to Claude Opus (most capable)
- Priority access during peak hours
- Projects, extended context
- Required for Claude Code

Recommendation: Start free. If you hit usage limits while doing real work, upgrade then — not before. The free tier handles learning perfectly.

Tool 2 — LM Studio — local AI with no terminal

The **Discover** tab — a built-in model browser:

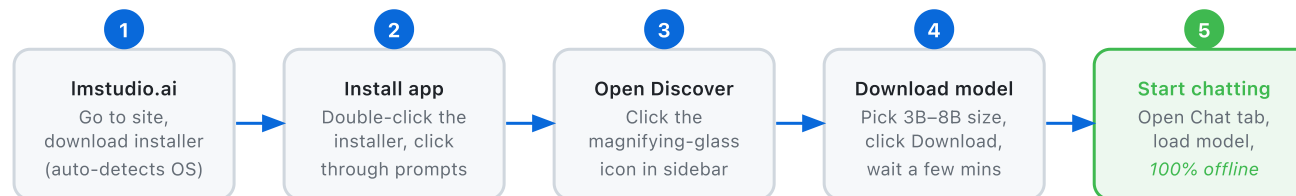


1. Download from **lmstudio.ai** — Mac, Windows, or Linux. Installs like any normal app.

2. Open the **Discover** tab. Find a 3B–8B model. Click *Download*.

3. Open **Chat**, load the model, start talking — *fully offline*.

LM Studio — install to first chat in 5 steps



Bonus — local API server: LM Studio can serve a local API on port `1234` (OpenAI-compatible format). Apps that normally call a cloud API can point at your machine instead — zero cost, zero data leaving home. You don't need this on day one; it's there in Module 6.

LM Studio vs Ollama — which should you pick?

LM Studio

- Point-and-click GUI — no terminal
- Built-in model browser with size warnings
- Visual chat interface just like Claude.ai
- Local API server toggle (one click)
- Best for: **beginners, visual people, anyone who wants no command-line**

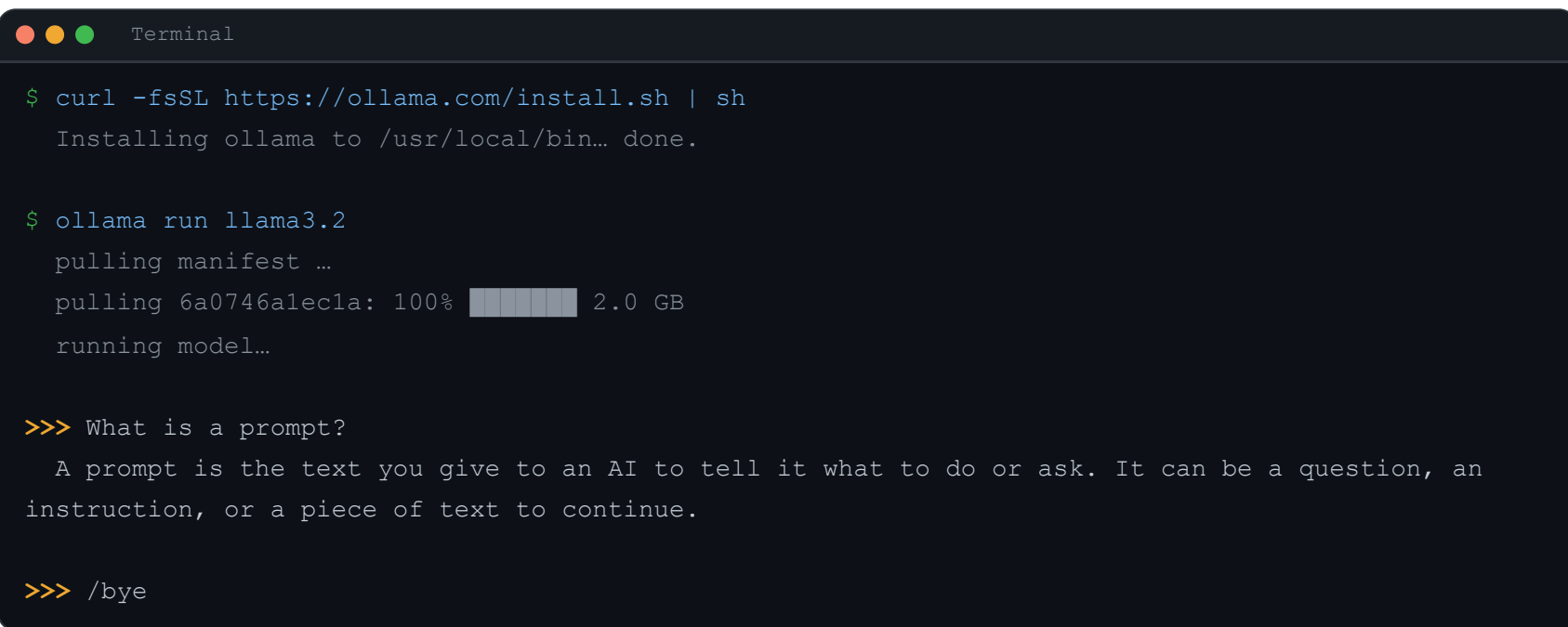
Ollama

- Tiny (~60 MB) — runs in the background
- One-command install & model download
- Scriptable: automate with a shell loop
- Local API on port 11434 (OpenAI-compatible.)
- Best for: **developers, servers, automation**

The one-line rule: LM Studio is friendlier (clicks, no terminal). Ollama is leaner and better for scripting. Most beginners try LM Studio first, then discover Ollama when they want to automate something.

Tool 3 — Ollama — AI in your terminal

Install once, then run any open model with a single command. First run downloads the model (a few GB); after that it loads in seconds.

A terminal window with a dark background and light text. The title bar shows three colored circles (red, yellow, green) and the word "Terminal". The terminal content shows the installation of Ollama using curl, followed by running the llama3.2 model. It shows progress bars for pulling the manifest and the model itself (2.0 GB). Finally, it shows a prompt being asked and answered, and then the user typing /bye to exit.

```
$ curl -fsSL https://ollama.com/install.sh | sh
Installing ollama to /usr/local/bin... done.

$ ollama run llama3.2
pulling manifest ...
pulling 6a0746a1ec1a: 100% ██████████ 2.0 GB
running model...

>>> What is a prompt?
A prompt is the text you give to an AI to tell it what to do or ask. It can be a question, an
instruction, or a piece of text to continue.

>>> /bye
```

Type `/bye` to exit. On Linux use the one-liner above. On Mac/Windows download from ollama.com/download.

Ollama — commands you'll actually use

Memorize these four and you'll cover 90% of what Ollama can do:

```
ollama run llama3.2      # download (first time) + start a chat
ollama list              # see every model you've downloaded
ollama pull mistral      # download a model without running it
ollama rm llama3.2       # delete a model to free up disk space
ollama serve             # run Ollama as a background API server
```

Popular starter models:

`llama3.2` — Meta's fast 3B model

`mistral` — Mistral 7B, excellent balance

`phi3` — Microsoft's small but smart model

Local API: Ollama automatically exposes an **OpenAI-compatible API on port 11434**. Your own scripts can call it just like the cloud API — free, no data leaving home. See Module 6.

Tool 4 — Claude Code — AI that acts, not just answers

Claude Code is a **coding agent**. You run it inside a project folder and describe what you want in plain English. It reads your files, writes or edits code, and runs commands on your behalf.

What it can do:

- Read your entire project codebase
- Write new files or edit existing ones
- Run terminal commands and see output
- Fix bugs you describe in plain English
- Scaffold a new project from scratch

What you need first:

- A paid Claude plan (Pro, Max, Team, or Enterprise)
- — or — an Anthropic Console account with billing
- macOS, Linux, or Windows (WSL)
- ~4 GB RAM (very light)

The free Claude.ai tier does not include Claude Code. If you're just starting out, set it aside and come back when you have a coding project to work on.

Claude Code — two install options

Option A — Native installer

Simplest. No extra software needed:

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

Works on macOS and Linux out of the box.

Option B — Via npm

If you already have Node.js 18 or newer:

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

On Windows: use inside WSL (Windows Subsystem for Linux).

After installing, `cd` into any project folder and type `claude`. It will open a browser window to authenticate on first run. That's it — you're running an AI coding agent.

Claude Code — one important rule before you start

Claude Code can edit real files and run real commands. It is not a chat window — it has actual access to your project on disk.

Do NOT start it on...

- A production system or live website
- A folder with files you haven't backed up
- Anything containing real credentials (.env)

DO start it on...

- A test or sandbox folder
- A project tracked by git (easy undo)
- A fresh directory for a new project

Think of it as a junior developer you're mentoring: talented, fast, but *you review the work before it goes live*.

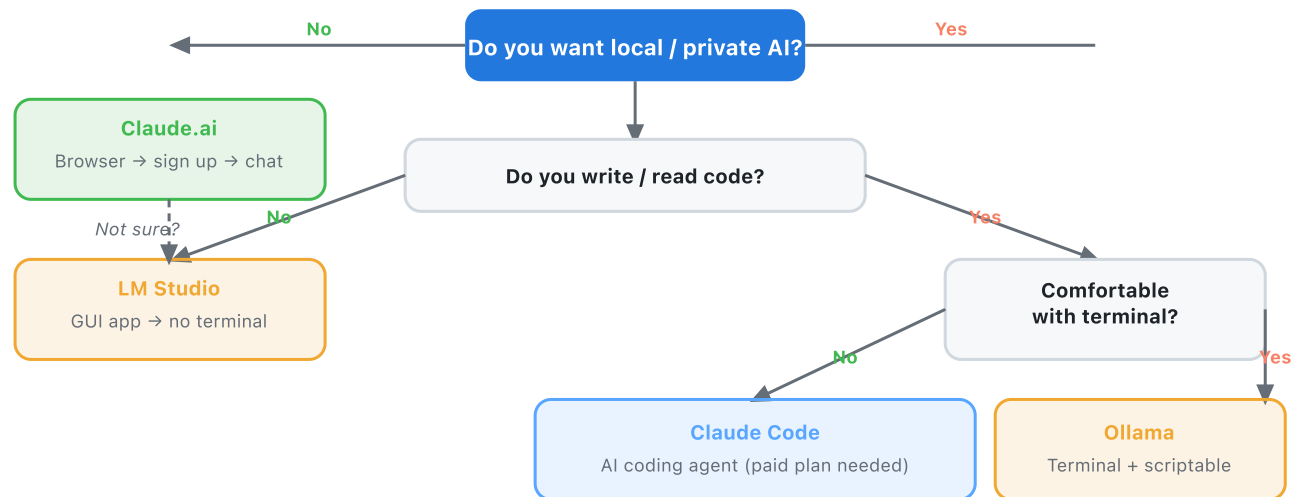
Read what it proposes before approving. More on agent safety in Module 9.

All four tools side by side

Tool	Runs where	Install?	Privacy	Best for
Claude.ai	Anthropic’s cloud	No	Data sent to Anthropic	Everyone, day one
LM Studio	Your machine (GUI)	App download	Fully offline	Private AI, no terminal
Ollama	Your machine (CLI)	Yes (one command)	Fully offline	Scripting, servers, devs
Claude Code	Your machine + cloud	Yes (curl or npm)	Code sent to Anthropic	Coding, multi-step tasks

All four install commands in one place: see the **Install Commands Gallery** in the Theory page — copy buttons included.

Pick your path — decision tree



Unsure about your situation? Use the **Tool Picker Wizard** in the Theory page — it asks 3 questions and gives you the exact install command.

A suggested 4-week path

Week 1 — Chat only. Use Claude.ai every day for real tasks. Learn what good output looks like. Practice asking — don't install anything else yet.

Week 2 — Go local. Install LM Studio, download a 3B–8B model, have the same conversation you had on Claude.ai. Compare quality.

Week 3 — Touch the API. Run the Python example from Module 6. Process a small batch of something you actually care about.

Week 4 — Build one tiny thing. Pick a starter project from Module 8 and finish it.

You don't need to rush. Each step works fine on its own. The later ones just unlock more. The single best thing you can do *today*: sign up for Claude.ai and use it for something real.

Recap — what you covered today

Four tools

- **Claude.ai** — cloud, zero install, start here
- **LM Studio** — local GUI app, offline & private
- **Ollama** — local terminal tool, scriptable
- **Claude Code** — AI coding agent, edits real files

Key install commands

```
# Ollama (Linux/WSL)
curl -fsSL https://ollama.com/install.sh | sh

# Claude Code
curl -fsSL https://claude.ai/install.sh | bash

# Or via npm
npm install -g @anthropic-ai/claude-code
```

Your homework

Install *one* tool. Use it for a real task. Take a screenshot for the homework check.

Lesson 05 →

Prompting — how to talk to AI well. The skill that improves your results more than any tool choice.

Installing the tools

Reading time: ~12 min · Prereqs: Lessons 01–03 · 3 interactive demos inside

In Lesson 03 we looked at the four *doorways* into AI: chat, API, CLI, and code tools. This lesson is where you actually walk through one of those doors. We'll cover four tools, from the absolutely-nothing-to-install option to a full AI coding agent you run from your terminal. You don't need all of them — you just need to pick the one that fits where you are right now.

The short version

- **Just want to use AI?** → claude.ai. Nothing to install. Skip to Section 1.
- **Want private, offline AI with a friendly interface?** → LM Studio. Section 2.
- **Comfortable in a terminal and want lean, scriptable local AI?** → Ollama. Section 3.
- **Want AI to help you write and edit code?** → Claude Code. Section 4.

Not sure which one? Use the wizard below before reading on.

Tool Picker Wizard

Answer three questions and we'll suggest the right starting point — with the exact install commands if needed.

Q1 — What's your operating system?

— pick one —



Q2 — How much RAM does your computer have?

— pick one —



Q3 — What do you want to do?

— pick one —



Find my tool

Reset

The recommendations below are starting points — you can always try more than one.

1. Claude.ai — the front door (nothing to install)

The simplest possible start with AI is also, frankly, the best one. Claude.ai is Anthropic's web interface for Claude — you open a browser, sign up, and start typing. That's the entire setup.

1. Go to claude.ai in any browser.
2. Sign up with email or a Google account.
3. Start typing. That's it.

There's a free tier that's more than enough for learning. Paid plans (Pro and Max) give you heavier use limits and access to the most capable models. There are also desktop and mobile apps if you'd rather have something in your dock or on your home screen — same login, same experience.

One important thing to understand: when you use Claude.ai, *your text travels to Anthropic's servers*. For most people and most tasks, that's completely fine. But if you're working with genuinely private information — medical records, confidential business data, anything you wouldn't email to a stranger — that's when the local options below become relevant.

Start here, no matter what

Even if you plan to run AI locally eventually, spend your first week with Claude.ai. You'll learn what good AI output looks like, what kinds of prompts work, and what you actually want to do — all without wrestling with setup. That context makes the local tools much easier to appreciate later.

2. LM Studio — local AI with a friendly app

LM Studio is a free desktop app for Mac, Windows, and Linux that lets you download and run open AI models *completely offline*, with a clean, chat-style interface. No command line, no coding, nothing scary. It's the gentlest possible way to try running AI on your own hardware.

Why would you want this? Two reasons: **privacy** (nothing leaves your machine) and **cost** (once a model is downloaded, running it is free — no API bills). The trade-off is quality: local models are good and improving fast, but they don't yet match the best cloud models on hard tasks. And they need RAM.

The hardware reality check

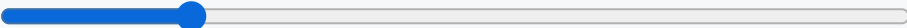
Here's the honest version. A model's size is measured in parameters (the internal numbers that store what the model learned). More parameters = smarter but heavier. The quick guide:

- **8 GB RAM:** small models (around 3B parameters). Works, but slowly. Good enough for simple questions and drafting.
- **16 GB RAM:** mid-size models (7B–13B). Comfortable for most everyday tasks.
- **32 GB+ RAM:** large models (30B+). Noticeably better quality; smooth performance.

A dedicated GPU — especially on an Apple Silicon Mac (M-series chip) — makes local models much faster. The M-series architecture shares memory between CPU and GPU, which is unusually good for running LLMs. If you have a modern Mac with an M chip, local AI is genuinely practical. On a modest Windows or Linux machine with no dedicated GPU, cloud AI is the better experience.

Hardware / Model-Size Matcher

Drag the slider to your RAM amount and see what you can comfortably run.

RAM:  16 GB

Small models (3B)

Mid models (7B–13B)

Large models (30B+)

8 GB RAM minimum
Slow but usable
~4–8 tokens/sec

16 GB RAM comfortable
Good for most tasks
~8–20 tokens/sec

32 GB+ RAM needed
Noticeably better quality
~20+ tokens/sec

At 16 GB RAM:

Largest comfortable model size: **Mid models (7B–13B params)**

Expected speed: **~8–20 tokens/sec**

16 GB is the sweet spot for most people. 7B–13B models like Llama 3.2 and Mistral 7B handle everyday writing, Q&A, summarisation, and simple code tasks well. Try `llama3.2` or `mistral`.

Tokens/sec estimates assume a modern CPU without a dedicated GPU. Apple Silicon and discrete GPUs can double or triple these numbers.

Installing LM Studio

1. Go to lmstudio.ai and download the version for your OS. It auto-detects.
2. Install it like any normal app and open it.
3. Open the **Discover** tab (magnifying-glass icon). This is a built-in browser of downloadable models. Each one shows its size and whether your hardware can handle it.
4. For a first model, pick something in the 3B–8B range. Click Download.
5. Once downloaded, open the **Chat** tab, load the model, and start chatting — fully offline, fully private.

Bonus: LM Studio as a local API server

LM Studio can run a local API server on port `1234` that mimics the standard OpenAI API format. That means apps which normally talk to a cloud AI can be pointed at your local one instead. You don't need this on day one, but it's there in Module 6 when you start writing code.

3. Ollama — local AI in the terminal

Ollama does the same job as LM Studio — download and run open models locally — but it lives entirely in the **terminal** (that black text window). If you've never used a terminal before, LM Studio is the friendlier path. But if you're comfortable typing commands, Ollama is faster to get going and much easier to automate.

Installing Ollama

Go to ollama.com/download and install for your OS. On Linux (or in WSL on Windows), you can use the one-line installer:

```
curl -fsSL https://ollama.com/install.sh | sh
```

Running your first model

Open your terminal (Terminal on macOS/Linux, PowerShell or Command Prompt on Windows, or a WSL window). Then type:

```
ollama run llama3.2
```

The first time, it downloads the model — a few gigabytes, so give it a minute. After that, it loads fast. You'll see a `>>>` prompt. Type your message and press Enter. Type `/bye` to exit.

The handy commands you'll actually use

```
ollama list          # see models you've downloaded
ollama pull mistral  # download a model without running it
ollama rm llama3.2   # delete a model to free up space
ollama serve         # run it as a background server
```

Like LM Studio, Ollama exposes a local API — by default on port `11434`, OpenAI-compatible. Module 6 shows how to use it.

LM Studio vs Ollama — the one-line summary

LM Studio is friendlier (clicks, no terminal). Ollama is leaner and better for scripting and servers. Most beginners try LM Studio first, then discover Ollama when they want to automate something.

4. Claude Code — an AI coding agent

Claude Code is a different kind of tool. It doesn't just *answer* — it *acts*. You run it inside any project folder on your computer, describe what you want in plain English, and it reads your files, writes or edits code, and runs commands on your behalf. It's what the previous lesson called a **code/agent tool**.

Because it can change real files, Claude Code is not the right first tool for a complete beginner with no coding background. But if you do write code — even a little — or if you're curious about where AI is heading, it's worth understanding.

What you need before installing

Claude Code requires a **paid Claude plan** (Pro, Max, Team, or Enterprise) or an **Anthropic Console account** with billing set up. The free Claude.ai tier doesn't include it. System requirements are light: macOS, Linux, or Windows, around 4 GB of RAM.

Two ways to install

Option A — Native installer (simplest, no extra software needed):

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

Option B — Via npm (if you already have Node.js 18 or newer):

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

On Windows, Option B works best inside WSL (Windows Subsystem for Linux).

After installing, navigate to any project folder in your terminal and type `claude`. It will ask you to authenticate in a browser window on first run. There are also extensions for VS Code and JetBrains if you'd rather use it inside your editor.

Use it on a test project first

Claude Code can edit real files and run real commands. Start it on a test folder or a project backed up with version control (git). Read what it proposes before approving. Think of it as a capable junior developer: you review the work before it goes live. More on agent safety in Module 9.

All four tools side by side

Tool	Runs where	Install?	Best for	Privacy
Claude.ai	Anthropic's cloud	No	Everyone, day one	Data leaves your machine
LM Studio	Your machine (GUI app)	App download	Private AI, no terminal	Fully offline
Ollama	Your machine (terminal)	Yes	Scripting, servers, developers	Fully offline
Claude Code	Your machine (terminal) + Anthropic's cloud	Yes	Coding, multi-step tasks	Code sent to Anthropic for processing

The install commands, all in one place

Use the cards below to copy the exact command for whichever tool you're installing.

Install commands

Click any Copy button to copy the command to your clipboard. Claude.ai requires no command — just a browser.

Claude.ai

No install needed. Open in any browser and sign up with email or Google.

LM Studio

Download the desktop app for Mac, Windows, or Linux. No command line needed.

```
https://claude.ai
```

```
https://lmstudio.ai
```

Ollama

One-liner installer for Linux (also works in WSL on Windows). Or download from ollama.com/download for Mac/Windows.

```
curl -fsSL https://ollama.com/install.sh | sh
```

Claude Code

Native installer (no extra dependencies). Requires a paid Claude plan or Anthropic Console account.

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

Commands are copied as plain text. For Ollama and Claude Code, paste into your terminal and press Enter.

Where to go from here

Now that you have at least one tool installed, the most important next step isn't learning another tool — it's learning *how to talk to AI well*. That's what Lesson 05 is about: prompting. The quality of your results depends far more on how you phrase your request than on which tool you use. A well-crafted prompt to a free model often beats a vague prompt to the most powerful one.

Before you head to Lesson 05, try the homework for this lesson — it's designed to get you actually using whichever tool you just set up.

LAST-REVIEWED: 2026-06-04

Homework — Installing the tools

Estimated time: ~45 min · Pick the track that matches what you installed

Two tasks, structured as tracks so you can focus on whatever you actually set up. Task 1 works for everyone — all you need is a browser. Task 2 is for anyone who installed LM Studio or Ollama and wants to kick the tyres on a local model. Do both if you can.

What you need

- **Task 1:** A browser and a claude.ai account (free tier is fine).
- **Task 2 (Track A):** [LM Studio](#) installed and at least one model downloaded.
- **Task 2 (Track B):** [Ollama](#) installed. No model needed in advance — the task downloads one.

If you haven't installed anything yet, use the [Tool Picker Wizard in the Theory page](#) to decide what to set up first.

Task 1 — A tour of Claude.ai

Even if you plan to use a local tool eventually, Claude.ai is the baseline. This task gives you a structured first visit so you learn the interface, not just stumble around it.

1. Go to claude.ai. Sign in (or sign up — the free tier covers everything in this task).
2. Start a new conversation. Type: **"I'm a complete beginner learning about AI. What are the three most useful things I can do with you in my first week?"**
Read the answer. Note whether it matches what the course has told you so far.
3. In the *same* conversation, follow up with one of the suggestions Claude just gave you. Ask it to actually do that thing — start a task, draft something,

explain something. The point is to use the tool for a real-ish purpose, not just ask it about itself.

4. Find the model selector. On `claude.ai` it's usually near the top of the chat window. Write down the model name you're talking to (e.g., "Claude Sonnet", "Claude Opus", etc.).
5. Start a **new** conversation (new chat, blank slate). Note that the new conversation has no memory of the first one. This is the context-window idea from Lesson 01 in action: every conversation starts fresh unless a tool adds memory.
6. In your notes: write two sentences — one thing the tool did that surprised you, and one thing you'd want to try next session.

✓ **Self-check: you've had a real back-and-forth conversation (not just one question); you know the model name; you've observed the "fresh start" behaviour of a new chat.**

Task 2 — Your first local model session

Pick the track that matches your setup. If you haven't installed a local tool yet, come back after Task 1 and installing one using the Theory page guide.

Track A — LM Studio

1. Open LM Studio. If you haven't downloaded a model yet, go to the **Discover** tab and find something in the 3B–8B range. Look for a model with a green "Fits in memory" indicator. Download it (a few gigabytes — give it time).
2. Open the **Chat** tab and load your model. You should see a chat interface that looks a lot like `Claude.ai`.
3. Ask the same question you asked Claude in Task 1: **"I'm a complete beginner learning about AI. What are the three most useful things I can do with you in my first week?"**
4. Compare the answers. In your notes: is the quality similar, noticeably worse, or surprisingly good? How long did the response take?
5. In LM Studio, look at the **model info panel** (usually on the right or in settings). Write down: the model name, the size in GB, and the parameter count (the "B" number). This is how you start reading model specs.

6. Close LM Studio. Notice: does the model keep running in the background, or does it fully unload? (Check your system's task manager or Activity Monitor.) This is relevant because large models can tie up your RAM even when you're not actively using them.

✓ **Self-check: you've run a complete chat session with a local model; you can name the model, its size, and rough parameter count; you've observed at least one quality or speed difference compared to Claude.ai.**

Track B — Ollama

1. Open your terminal (Terminal on macOS/Linux, PowerShell or a WSL window on Windows).
2. Run:

```
ollama run llama3.2
```

If this is the first time, Ollama will download the model (~2 GB). Wait for it to finish. You'll see a `>>>` prompt.

3. Type: **I'm a complete beginner learning about AI. What are the three most useful things I can do with you in my first week?** and press Enter.
4. Compare the answer to what Claude gave you in Task 1. Note the quality and how long it took. Also note: the whole session happened on your computer, with no internet needed after the download.
5. Try two more quick commands to get comfortable with the interface:

```
ollama list
```

```
ollama pull mistral
```

Note what each one shows or does. Type `/bye` to exit the llama3.2 chat first if you're still in it.

6. In your notes: write the model name, roughly how large the download was, and one thing the terminal interface does differently from a chat app with a graphical interface.

✓ **Self-check:** you've completed a chat session in the terminal; you've run `ollama list` and seen at least one model listed; you can describe one difference between a CLI tool and a GUI app in your own words.

Bonus — compare cloud vs local on the same hard question (~15 min)

Ask both Claude.ai and your local model the same difficult question — something where accuracy matters, like a historical fact, a calculation, or a reasoning puzzle.

Examples:

- *"A train travels 120 km at 80 km/h, then 60 km at 40 km/h. What is the average speed for the whole journey?"*
- *"What was the main reason the Roman Empire split in 285 AD?"*
- *"I have a list of 10 items. I want to pick 3 of them. How many different combinations are there?"*

Compare the answers. Does the local model match the cloud model? Does one sound more confident than the other? Do you notice any hallucination? Verify the right answer independently. This side-by-side comparison is the fastest way to build an intuition for when local models are good enough and when you need the cloud.

What "done" looks like

In your notes you should have:

- The model name you used on Claude.ai, and two sentences from your session (one surprise, one next-step idea).
- If you completed Task 2: the local model name, size in GB, and parameter count — and at least one quality or speed observation compared to the cloud.
- A one-sentence answer to: "In what situation would I prefer a local model over Claude.ai?"

That last question is the real takeaway. There's no universal right answer — it depends on your privacy needs, your hardware, your internet reliability, and what you're doing. The point is that you've now used both and can make an *informed* choice.

Lesson 05: How to talk to AI well (prompting)

Lesson 05 — How to talk to AI well

The skill that separates “AI is useless” from “AI is incredible.” The model is almost never the problem — the prompt usually is.

AI for Dummies

Module 5

Heavy AI users aren't gifted — they learned one habit: **give the AI everything it needs, right there in your message.**

The one rule that explains everything

Vague in → vague out Specific in → useful out.

✗ VAGUE PROMPT

"recycling lesson ideas"

What you get: a wall of generic bullets that could suit any age, any duration, any classroom in the world. You spend 30 minutes adapting it.

✓ SPECIFIC PROMPT

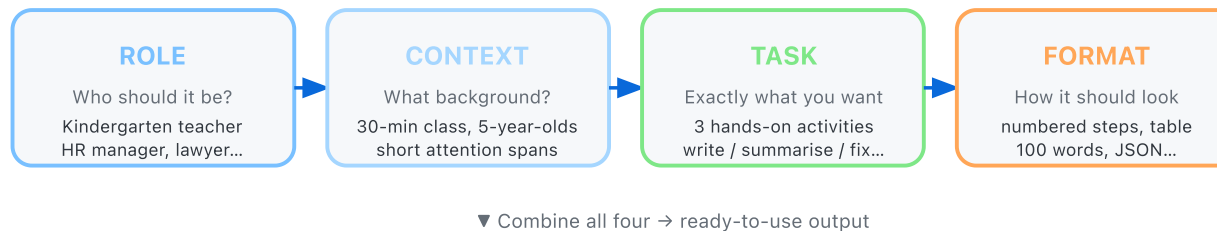
"You are an experienced kindergarten teacher. I need three hands-on recycling activities for 5-year-olds, 30 min total. List each as a numbered step with materials and timing."

What you get: three print-ready activities tuned exactly to your class. 2 minutes of effort.

Key insight: The AI has no memory of you and no idea what "good" means to you. It only knows what is in your message right now. Give it everything it needs.

The recipe: Role → Context → Task → Format

Four ingredients. Every strong prompt has at least three. All four = near-perfect output.



You don't need all four every time. A quick casual question can survive on Task alone. But the more the output matters, the more ingredients you should add. The sweet spot for real work is all four.

R — Role: put it in the right frame of mind

The Role ingredient sets the perspective, vocabulary, and depth of response. It's the first thing to add when results feel generic.

EXAMPLE ROLES

"You are an experienced kindergarten teacher."

"You are a plain-English solicitor."

"You are a senior data analyst who explains findings to non-technical executives."

Why Role works

Without a role, the AI defaults to a generic helpful voice — useful for no one in particular. Giving a role forces it to adopt the right vocabulary level, the right assumptions, and the right perspective.

"You are a nutritionist" gets different (better) meal advice than no role at all.

The role can be a job title, an expertise level, or a perspective: *"You are a sceptical editor."*

C — Context: give it the background it can't guess

The AI can't ask follow-up questions the way a colleague would. Give it the context *upfront*.

"I'm planning a 30-minute lesson for 5-year-olds about recycling. They have short attention spans and learn best through movement and colour."

"Our SaaS product had a 4-hour outage yesterday. About 2,000 customers were affected. We've fixed the root cause."

"I'm a non-technical finance manager. I need to brief my CEO on blockchain in 2 minutes at tomorrow's board meeting."

What to include in Context

- Who you are and what you're working on
- Who the audience is (age, expertise, role)
- Any relevant constraints (time, budget, length)
- What you already know or have tried
- Any background facts the AI needs to use

Rule of thumb: If a smart new colleague would need to ask you that question on day one, put the answer in Context.

T — Task: be specific about what you want

Vague tasks produce vague output. The Task ingredient specifies *scope*, *quantity*, and *nature* of the output.

✗ VAGUE TASK

"give me ideas"

The AI will produce a wide, shallow list. Could be 3 ideas, could be 20. Could be high-level, could be tactical. You don't know what you'll get.

✓ SPECIFIC TASK

"Give me three hands-on activities."

Quantity: three. Nature: hands-on. Scope: activities. The AI knows exactly what to produce.

Task precision checklist

- **Verb** — write, summarise, compare, explain, fix, translate, list...
- **Quantity** — three ideas, one paragraph, five bullets...
- **Scope** — about X, focusing on Y, limited to Z...
- **Audience** — for a beginner, for my manager, for a press release...

The more of these you specify, the tighter the output.

F — Format: tell it how the answer should look

Without format guidance, the AI defaults to flowing prose paragraphs. **That's rarely the most useful shape.**

"List each as a numbered step with the materials needed and the time it takes."

"Give the answer as a table with columns:
Option | Pros | Cons | Cost."

"100 words max. Plain language. No jargon."

"Format as an email: subject line, 3 short paragraphs, friendly sign-off."

Format options cheat sheet

- **Table** — comparisons, options, specs
 - **Numbered list** — instructions, steps
 - **Bullet list** — quick scanning
 - **JSON / code** — structured data
 - **Word limit** — "50 words max"
 - **Document sections** — intro / body / summary
 - **Email / report / tweet** — named templates
-

All four pieces — assembled into one prompt

This is what a complete, ready-to-paste prompt looks like. Each colour is one ingredient.

You are an experienced kindergarten teacher.

I'm planning a 30-minute lesson for 5-year-olds about recycling. They have short attention spans and learn best through movement and colour.

Give me three hands-on activities.

List each as a numbered step with: the materials needed, the time it takes, and one tip for keeping 5-year-olds engaged. Keep each description under 70 words.

● Role ● Context ● Task ● Format

What you get back: three print-ready activities, each labelled, timed, and adapted for 5-year-olds. No editing required. Copy straight into your lesson plan.

Vague vs. Specific — the AI's actual responses

✗ PROMPT: "RECYCLING LESSON IDEAS"

AI RESPONSE (mock)

Here are some recycling lesson ideas:

- Teach the three Rs: Reduce, Reuse, Recycle.
- Have students sort items into recycling categories.
- Show a video about recycling and discuss.
- Create art projects using recycled materials.
- Discuss why recycling matters for the environment.

These activities can be adapted for any age group.

✓ PROMPT: THE FULL FOUR-PART RECIPE

AI RESPONSE (mock)

1. **The Sorting Game (8 min)**

Materials: box of mixed items, 3 labelled bins.

Each child picks one item and places it in the right bin. Ask "why?" after each. *Tip: use bright, oversized labels.*

2. **Bottle-Cap Mosaic (12 min)**

Materials: bottle caps, glue sticks, card stock.

Kids arrange caps into a shape. *Tip: pre-draw outlines for struggling children.*

3. **Paper-to-Paper Circle (10 min)**

Materials: newspaper, water, mesh screen, sponge. Demonstrate recycled paper; kids press pulp themselves. *Tip: let them feel the wet pulp first.*

Same topic. Wildly different utility. The vague response could have been written by anyone, for anyone. The specific response was written for *you*, for your class, right now.

Habit 1 — Show an example (the most underused trick)

Words like “catchy,” “professional,” or “friendly” mean different things to different people. **An example is unambiguous.**

✗ WITHOUT EXAMPLE

"Write a catchy subject line for our product launch."

The AI guesses what "catchy" means to you. Could be punny. Could be urgent. Could be click-bait. You get a lottery ticket.

✓ WITH EXAMPLE

"Write a subject line like this one: 'Your invoice is ready – 3 things to check.' Now write one for our product launch on Tuesday."

The AI matches your style, tone, and structure exactly.

Where to use examples

- Email subject lines & body copy
- Job ad or social-media post style
- Report or document formatting
- Code style ("write functions like this one")
- Tone of voice ("write in the style of this paragraph")

Why it works: Pattern-matching is what LLMs are best at. Give them a pattern to match.

Habit 2 — Ask for the format you want

AI defaults to paragraphs. **Paragraphs aren't always what you need.** Say so — it complies every time.

"Give the answer as a table with columns:
Tool | Free? | Best for | Biggest downside."

"Format as a numbered step-by-step guide. No
prose. Steps only."

"50 words maximum. Plain English. No sub-
headings."

"Format as a tweet thread — 5 tweets, each
under 280 characters, numbered."

Why explicit format matters

The AI has no idea whether your output will be pasted into a spreadsheet, read aloud, or put on a slide. You do. **Tell it.**

Format guidance also caps length. A prompt without a word limit will often produce more than you need.

Quick win: Add *"Give your answer as a table"* to any comparison question and the result becomes immediately scannable.

Habit 3 — Tell it what *not* to do

Positive instructions alone sometimes aren't enough. A "don't" constraint closes the bad paths before the AI wanders down them.

"Don't use jargon or technical terms."

"Don't make anything up — if you're unsure, say so."

"Don't suggest anything that requires a paid subscription."

"Don't start the email with 'I hope this message finds you well'."

"Don't give me more than five options."

The hallucination stopper

The single most useful "don't" for factual questions:

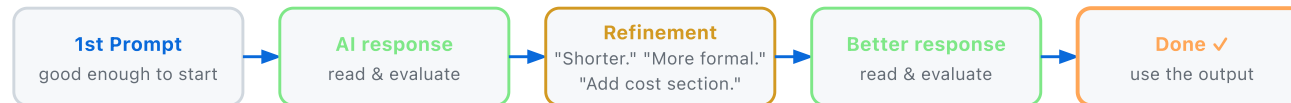
"If you don't know, say you don't know. Don't guess."

This dramatically reduces hallucinations because it makes the model flag uncertainty rather than filling gaps with confident-sounding nonsense.

Use "don't" for style, scope, and facts. Think of it as building a fence around the output space.

Habit 4 — Iterate, don't restart

Your first prompt is a first draft, not a final request. Stay in the same chat and steer.



Refinement commands that always work

- `"Shorter."` — cuts length
- `"More formal."` — adjusts tone
- `"Put that in a table."` — reformats
- `"Now add a section on cost."` — extends
- `"Remove bullet 3."` — trims

The chat history carries all context. You don't repeat yourself — you build on what's already there. This is faster than starting over *every single time*.

Habit 4 in action — watch the refinement loop

▶ Replay

Three small messages in the same conversation. No rewriting from scratch. Each step builds on the last.

Habit 5 — Let it think for hard problems

Adding “*think step by step*” genuinely improves accuracy on reasoning tasks. It’s the equivalent of asking someone to show their working.

✗ WITHOUT STEP-BY-STEP

Prompt: "If a train leaves at 09:15 and takes 2h 45min, plus a 20-minute connection, what time does it arrive?"

AI response (mock): It arrives at 12:20.

Wrong. The AI skipped the addition and landed on a plausible-sounding answer.

✓ WITH “THINK STEP BY STEP”

Prompt: "...Think step by step."

AI response (mock):

1. Depart 09:15.
2. Add 2h 45min → 12:00.
3. Add 20-min connection → 12:20.
4. Final arrival: **12:20.**

Each step is visible. Errors are easy to spot and correct.

Phrases that trigger step-by-step thinking:

"Think step by step." · "Show your reasoning." · "Walk me through your logic."

When to use it: logic puzzles, multi-step maths, planning decisions, comparing options. Don't use it for simple tasks — it adds unnecessary length.

Common mistake 1 — Being too vague

✗ THE MISTAKE

"help me write something"

"give me ideas"

"explain blockchain"

These prompts work, but they return generic outputs that could apply to anyone, anywhere. You spend more time adapting than you saved by asking.

✓ THE FIX

Add Role + Context + Task. Even Role + Task doubles quality. Even one sentence of context narrows the output dramatically.

"Explain blockchain to a non-technical finance manager who needs to brief their CEO in 2 minutes."

Why vagueness is expensive

When the prompt is vague, the AI has to guess. It guesses middle-of-the-road. Middle-of-the-road output for your specific situation is almost never good enough to use without heavy editing.

Specificity is the cheapest form of editing. Five extra seconds writing the prompt saves five minutes fixing the output.

Rule: If a stranger reading your prompt couldn't tell what a "good" answer looks like, the AI can't either.

Common mistake 2 — Ten things in one prompt

✗ THE MISTAKE

```
"Summarise this article, then rewrite it in a friendlier tone, then translate it to French, then suggest a tweet thread, then give me a 10-word headline, then check it for factual errors."
```

The AI tries to do everything at once and cuts corners on each. The summary is thin, the rewrite is rushed, the translation is mechanical.

✓ The fix: one task per message

Chain the tasks sequentially in conversation instead:

1. Message 1: "Summarise this article."
2. Message 2: "Now rewrite it in a friendlier tone."
3. Message 3: "Translate that to French."

Each task gets full attention. Each output is complete. The iteration habit (Habit 4) handles the rest.

One clear exception: you can ask for multiple *related items of the same type* in one prompt — e.g., "give me five subject line options" is fine. The rule is one *task type*, not one item.

Common mistake 3 — Trusting the first answer on facts

Warning: AI can generate false information with complete confidence. A statistic, a quote, a date, a job title — these can all be *wrong* even when the AI sounds certain. This is called a **hallucination**.

What hallucination looks like

You ask: *"What did Einstein say about imagination?"*

The AI produces a confident-sounding quote with a citation. The quote sounds exactly right. **It may be entirely fabricated.**

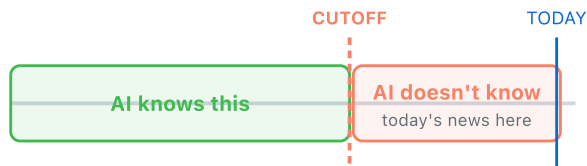
Higher risk with: niche topics, specific numbers, recent events, named people.

The fix: verify anything that matters

- Never cite an AI-generated fact without checking it.
 - Add *"if you're not sure, say so"* to factual prompts.
 - Ask for sources — then actually look them up.
 - Treat AI output as a first draft to be verified, not a final answer.
-

Common mistake 4 — Forgetting the knowledge cutoff

Most AI models stopped learning at a specific date — usually six months to a year or more before you're talking to them. **Anything after that date is unknown to the model.**



✗ WILL PRODUCE WRONG ANSWERS

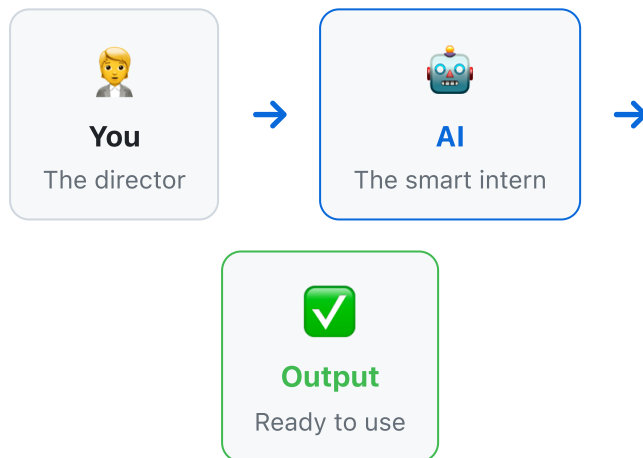
"What is the current price of Bitcoin?"
"Who won the 2025 election?"
"What are the latest AI models available?"

✓ THE FIX

Use a tool with web search (Perplexity, Claude with web browsing, ChatGPT browsing mode).
Or: paste the current information directly into your prompt and ask the AI to work from what you've given it.

The mental model: AI as a smart intern

Think of the AI as a brilliant generalist who just joined your team **this morning** and knows nothing about your project, your company, or your preferences.



What this means in practice

- The intern is *capable of almost anything* — but needs direction.
- A vague “handle this” gets a generic result.
- A clear brief gets exactly what you need.
- You don’t blame the intern for not reading your mind.
- Good directors give specific notes — they don’t just say “be better.”

The shift: You are not a user of a search engine. You are the director of a capable assistant. Your job is to give a good brief.

Try it now — build a real prompt

Pick a task you actually have this week. Write a four-part prompt for it right now.

You are [your role here].

I am [who you are].

I need [the background they need to know].

[Any constraints or audience info].

[Exactly what you want — verb + scope + quantity].

[How you want the answer: table / list / email / word limit / format].

Don't [anything you want excluded].

● Role ● Context ● Task ● Format ● Negation

Starter ideas if you're stuck

- An email you've been putting off writing
- A confusing document you need summarised
- A decision you want to think through
- A meeting agenda for next week
- A job ad, report, or proposal draft

After you get the first response, use **one refinement** — "Shorter," "More formal," or "Add X."
Notice how fast you reach a usable result.

Your pocket reference — the whole lesson on one slide

Ingredient / Habit	What it does	Quick example
Role	Sets perspective & vocabulary	<i>"You are a plain-English lawyer."</i>
Context	Gives necessary background	<i>"I'm reviewing a rental contract for the first time."</i>
Task	Specifies exactly what you want	<i>"Flag any clauses that could cost me money."</i>
Format	Controls output shape	<i>"Bullet list, plain English, under 150 words."</i>
Negation	Closes bad paths	<i>"Don't use jargon. Don't make anything up."</i>
Show example	Pattern-matches style exactly	<i>"Write it like this: [paste example]."</i>
Iterate	Steers without restarting	<i>"Shorter." / "More formal."</i>

Ingredient / Habit	What it does	Quick example
Think step by step	Improves reasoning accuracy	<i>Add to any logic / maths / planning prompt.</i>

When a better prompt isn't the answer

Prompting is powerful, but it can't give a model information it doesn't have.



Current information

The model's knowledge stopped at a cutoff date. No prompt can conjure today's news. **Fix:** use a tool with web search (Perplexity, Claude with web mode).



Precise arithmetic

LLMs reason about maths better than they calculate it. For number-critical work, use a model with a code interpreter or a real calculator. **Always verify numbers.**



Your private documents

The model can't see files it hasn't been given. **Fix:** paste the relevant section, or use a tool with file upload / document Q&A.

Bottom line: prompting is the steering wheel. But you still need to know what the car can and can't do. Know your tool's limits.

What you now know — and what's next

This lesson in one paragraph

The four-part recipe (Role + Context + Task + Format) is your starting point. The five habits (show an example; ask for the format; say what not to do; iterate; let it think) sharpen it. The four mistakes (vague, overloaded, unchecked, outdated) are what to avoid. Treat the AI like a smart intern who needs a clear brief, not a search engine you query with three words.

The skill that compounds: every other AI task in this course gets faster once prompting clicks. This lesson earns its “most important” status.

Theory page →

Interactive prompt builder · Vague vs. specific toggle · Animated iteration demo · Mistakes checklist. Build and test your own prompts right there.

[theory.html →](#)

Homework →

Write real four-part prompts for three scenarios from your own life, run them, and reflect on what worked. The only way to get good is to practise.

[homework.html →](#)

How to talk to AI well

Reading time: ~15 min · Prereqs: Lessons 01–04 · 4 interactive demos inside · This is the lesson that changes everything.

Why this matters more than any other lesson

The single most common complaint about AI tools is "I tried it and it gave me useless answers." Almost every time, the problem is not the AI. The problem is the prompt. A well-crafted prompt to a mediocre model often beats a lazy prompt to the best model on the market.

This lesson teaches you *prompting* — the art of writing messages to AI in a way that gets you useful, specific, ready-to-use output. It's the skill that compounds. Once it clicks, every other AI tool you ever use gets better.

The core idea: AI can't read your mind

Here's the fundamental truth: **the AI has no idea who you are, what you're working on, or what "good" looks like to you.** It has no memory of you from a previous session (unless a tool specifically adds that). It only knows what's in your current message.

That sounds like a limitation. It's actually an instruction. *Give it everything it needs to help you well, right there in your message.* The more relevant context and direction you include, the better the result.

The one rule that explains everything

Vague in → vague out. Specific in → useful out.

This isn't a criticism of AI. It's how all communication works. If you tell a colleague "help me with the thing," you'll get a puzzled look. If you say "I need a five-bullet

summary of this document for a 10-minute meeting with the finance team," you'll get something useful. AI is the same.

A reliable recipe: Role, Context, Task, Format

Researchers and heavy users have converged on a simple structure that works across almost any task. Think of it as the four ingredients of a useful prompt:

- **Role** — who the AI should act as. *"You are an experienced kindergarten teacher."* This puts it in the right frame of mind, vocabulary, and perspective.
- **Context** — the background it needs to help you well. *"I'm planning a 30-minute lesson for 5-year-olds about recycling. They have short attention spans."* The AI can't ask follow-up questions like a human colleague would. Give it the context upfront.
- **Task** — exactly what you want it to do. *"Give me three hands-on activities."* Be specific about scope, quantity, and nature of the output.
- **Format** — how you want the answer structured. *"List each as a numbered step with the materials needed and the time it takes."* Without format guidance, the AI defaults to prose paragraphs, which aren't always what you need.

You don't always need all four. A quick casual question can get by with just the Task. But the more the output matters, the more ingredients you should include. The sweet spot for most work tasks is all four.

INTERACTIVE 1 OF 4 — BUILD YOUR PROMPT

Role

Who should the AI act as?

You are an experienced kindergarten teacher.

Context

What background does it need?

I'm planning a 30-minute lesson for 5-year-olds about recycling. They have short attention spans.

Task

Exactly what do you want?

Format

How should the answer look?

Give me three hands-on activities.

List each as a numbered step with materials and time needed.

ASSEMBLED PROMPT

Start filling in the fields above and your prompt will appear here...

Copy assembled prompt

Example: Recycling lesson

Example: Email rewrite

Clear all

● Role ● Context ● Task ● Format

Build your prompt piece by piece. Copy it and paste directly into Claude, ChatGPT, or any AI chat.

Seeing the difference: vague vs. specific

Reading about the recipe is one thing. Seeing the contrast in action is another. Below are two prompts for the exact same task — one vague, one using the four-part recipe. Toggle between them to feel the gap in what you get back.

INTERACTIVE 2 OF 4 — VAGUE VS. SPECIFIC

WITHOUT THE RECIPE

"recycling lesson ideas"

AI RESPONSE (MOCK)

Here are some recycling lesson ideas:

- Teach children about the three Rs: Reduce, Reuse, Recycle.
- Have students sort items into recycling categories.
- Show a video about recycling and discuss.

WITH THE RECIPE

"You are an experienced kindergarten teacher. I'm planning a 30-minute lesson for 5-year-olds about recycling. They have short attention spans. Give me three hands-on activities. List each as a numbered step with the materials needed and the time it takes."

AI RESPONSE (MOCK)

- Create art projects using recycled materials.
- Discuss why recycling is important for the environment.

These activities can be adapted for any age group.

1. The Sorting Game (8 min)

Materials: a box of mixed items (paper, plastic bottle, apple core, tin can), three labelled bins.

Each child picks one item and places it in the right bin. Ask "why?" after each choice.

2. Bottle-Cap Mosaic (12 min)

Materials: collected bottle caps, glue sticks, card stock.

Kids arrange caps into a rainbow or animal shape — takes fine motor, no scissors needed.

3. The Paper-to-Paper Circle (10 min)

Materials: newspaper, water, a blender (teacher only), a mesh screen, sponge.

Demonstrate making recycled paper; children press the pulp onto the screen themselves.

Highlight the specific prompt instead

Both prompts ask for "recycling lesson ideas." One gets a generic list; the other gets something you could print and use today.

Five habits that instantly improve results

The four-part recipe is your starting point. These five habits take you further, and they work alongside the recipe rather than replacing it.

1. Show an example

This is the single most underused trick in prompting. Words like "professional," "friendly," or "concise" mean different things to different people. An example is unambiguous.

Add a line like: "Write it like this: [paste your example here]." The AI will match the pattern, tone, length, and structure of the sample you provide. Use it for emails, social

posts, job ads, reports — anything where the style matters.

2. Ask for the format you want

Left to its own devices, the AI defaults to flowing prose. That's often not what you need. Be explicit: "Give the answer as a table with columns for X, Y, Z." Or "100 words max." Or "Format this as a bulleted FAQ." The AI will comply, and the result will be far more useful.

3. Tell it what *not* to do

Positive instructions alone sometimes aren't enough. Adding a "don't" constraint closes off the paths you don't want:

- "Don't use jargon or technical terms."
- "Don't make anything up — if you're unsure, say so."
- "Don't suggest paid tools."
- "Don't start the email with a pleasantry."

That third one — "don't make anything up" — is worth using any time you ask a factual question. It doesn't eliminate hallucinations entirely, but it dramatically reduces them because the model becomes more likely to flag uncertainty rather than paper over it with confident-sounding nonsense.

4. Iterate, don't restart

New users tend to rewrite their entire prompt from scratch when the first response isn't quite right. That's the slow path. The fast path is to stay in the same conversation and steer:

- "Shorter." — cuts length.
- "More formal." — adjusts tone.
- "Put that in a table instead." — reformats.
- "Now add a section on cost." — extends.

This is exactly like editing a document with a colleague. You don't throw the document away because one paragraph is off — you revise that paragraph. The chat history gives the AI all the context from your earlier turns, so you can build on it rather than repeating yourself.

The animation below shows a real iteration sequence. Hit **Replay** to watch it again.

INTERACTIVE 3 OF 4 — ITERATION IN ACTION

Replay animation

Three small refinements in the same conversation. No rewriting from scratch. Each step builds on the last.

5. Let it think for hard problems

For anything that involves multi-step reasoning — planning, comparing options, solving logic puzzles, working through a problem — add one of these phrases to your prompt:

- *"Think step by step."*
- *"Show your reasoning."*
- *"Walk me through how you arrived at that."*

When the model has to write out each step of its reasoning, it makes fewer errors. It's the equivalent of asking someone to show their working in a maths exam. The act of externalising the process keeps the thinking more honest. Don't use it for simple tasks — it makes those unnecessarily verbose. Save it for genuinely complex questions.

The "one colleague" mental model

Think of the AI as a brilliant generalist who just joined your team this morning and knows nothing about your project, your company, or your preferences. They're capable of almost anything — but they need direction. What would you tell a smart new colleague on their first hour to help them help you? Write that.

Common beginner mistakes (and how to fix them)

You don't have to learn these the hard way. The checklist below covers the five most common traps. Check each one off as you read it — your progress is saved between visits.

INTERACTIVE 4 OF 4 — KNOW THE MISTAKES BEFORE YOU MAKE THEM

0 / 5 read

☐ Being too vague

☐ Asking ten things in one prompt

☐ Trusting the first answer on facts and numbers

☐ Giving no context about who you are or what you need

☐ Forgetting the model doesn't know today's news

Click any item to expand the explanation. Your progress is saved so you can pick up where you left off.

When a better prompt isn't the answer

Prompting is powerful, but it's not magic. A few things require more than better wording:

- **Genuinely current information** — if you need today's news, currency rates, or live data, use a tool with web search. No prompt can give an offline model information it doesn't have.
- **Your private documents** — the model can't see files it hasn't been given. Paste the relevant section into your message, or use a tool that supports file uploads.
- **Precise maths** — language models are better at reasoning about maths than at doing arithmetic with guaranteed accuracy. For number-critical work, use a tool with a code interpreter that can execute calculations exactly.

One habit to start today

Next time you get a disappointing AI response, resist the urge to give up. Instead, look at your prompt and ask: *Which of the four parts (Role, Context, Task,*

Format) was I missing? Add the missing piece and try again. Nine times out of ten, the second attempt is dramatically better.

Summary

Prompting is a learnable skill, not a talent. The four-part recipe (Role + Context + Task + Format), the five habits (show an example; ask for the format; say what not to do; iterate; let it think step by step), and awareness of the five common mistakes — that's the whole toolkit. It fits on a single index card. The only thing left to do is practise.

The homework page gives you three scenarios to work through. Each one asks you to write a real prompt, run it, and reflect on what came back. That practice loop is where the skill actually builds.

Homework — How to talk to AI well

Estimated time: ~45 min · All you need: access to Claude, ChatGPT, or any AI chat · No code, no installs

The only way to build prompting skill is to practise. These three tasks ask you to write real four-part prompts, run them in an actual AI tool, and reflect on what came back. Treat each task as a genuine experiment, not a test with one right answer.

Before you start

Open Claude (claude.ai), ChatGPT (chat.openai.com), or any AI chat you have access to. You'll be pasting your prompts in there and looking at what comes back. Keep the theory page open in another tab so you can reference the four-part recipe as you go.

Task 1 — The four-part prompt: a task that's real to you

Pick a task from your actual life or work — something you genuinely need done or have struggled to get good AI output on before. It could be drafting an email, planning an event, writing a bio, summarising a document, preparing for a conversation, or anything else.

Step A: Write a vague prompt first. Just a few words. Run it and note what you get.

Step B: Now write a four-part prompt for the same task. Use the recipe:

- **Role:** who the AI should act as.
- **Context:** the background it needs. What's the situation? Who's involved? What do you already know?
- **Task:** exactly what you want it to do. Be specific about scope and quantity.

- **Format:** how you want the answer. List? Table? Paragraph? Word count? Tone?

Run it. Compare the two responses.

Step C: Iterate at least twice in the same chat. Use one-line refinements ("shorter", "more formal", "add a section on X") to improve the output further.

Reflect and write down:

- What was the vague prompt? What did it produce?
- What was the four-part prompt? Paste it here.
- What changed? What was noticeably better in the second response?
- What two refinements did you make? Did the third or fourth version feel close to usable?

✓ **Self-check: You should have two clearly different AI responses for the same underlying task, plus at least two refinement steps visible in your chat history.**

Task 2 — Show an example and use negative constraints

This task focuses on Habit 1 (show an example) and Habit 3 (tell it what not to do) — two of the most powerful but least-used techniques.

The scenario: You need to write a short introduction for a speaker at an event (or adapt this to your own context: a product blurb, a job posting, a social-media bio — any short piece where style and tone matter).

Step A: Find or write one example of an introduction you like. It doesn't have to be long — two to four sentences is fine. This is your style reference.

Step B: Write a prompt that includes:

- The example you found (paste it in after "Write it in the style of this example:")
- Relevant context about the speaker (or product, role, etc.)
- At least two "don't" constraints, e.g. "Don't use jargon," "Don't start with 'I am pleased to introduce'," "Don't be longer than 80 words."

Run it and review the output.

Step C: Remove the example from your prompt and run it again with everything else the same. Compare.

Reflect and write down:

- Paste both prompts and both responses.
- Did the example meaningfully change the output's style? In what way?
- Did the "don't" constraints work? Were any ignored?
- What did you learn about how negative constraints interact with positive instructions?

✓ **Self-check:** You should have two responses — one with an example, one without — that differ noticeably in style or structure. You should also be able to point to at least one place where a "don't" constraint had a visible effect on the output.

Task 3 — The step-by-step technique on a tricky question

This task tests Habit 5: letting the AI think for hard problems by asking it to show its reasoning.

Pick one of these scenarios (or substitute your own similarly complex question):

- You're deciding between two job offers with different salaries, benefits, commutes, and growth potential. Ask the AI to help you think through the decision.
- You're planning a trip to a city you've never visited and want to optimise for 3 days, limited budget, and a specific interest (food, museums, street art — your choice).
- You're trying to understand a clause in a contract or rental agreement that confuses you. (Invent or paraphrase a clause if you don't have a real one.)

Step A: Write a prompt *without* "think step by step." Run it.

Step B: Add "Think step by step" (or "Show your reasoning") to the end of the same prompt. Run it.

Step C: Iterate once: ask a follow-up like "Now give me just the final recommendation in two sentences" to extract a clean conclusion from the reasoning.

Reflect and write down:

- What question did you choose, and why?
- Paste both versions of the prompt and their responses.
- Did the "step by step" version surface anything the first version missed? Did the structure of the reasoning feel more trustworthy?
- Was the final two-sentence summary useful? Did iterating in the same conversation feel natural?

✓ **Self-check:** You should have two responses to the same question — one with, one without step-by-step reasoning — plus a short follow-up refinement. The reasoning version should visibly break the problem into stages rather than jumping straight to a conclusion.

Task 4 (Bonus) — Build your personal prompt cheat sheet

After completing the three tasks above, you have enough experience to start building your own reference. This bonus task asks you to synthesise what you've learned into something you'll actually use.

Write a personal prompt cheat sheet. It should cover:

- The four-part recipe, with one worked example from your own life (not the recycling example from the lesson — use your own).
- Two or three "don't" constraints that are useful to you personally.
- One example you like that you could re-use as a style reference for future prompts.
- Any pattern you noticed in your homework that you want to remember.

Keep it short — half a page or a few bullet points. You can write it here, in a note-taking app, or on paper. The format doesn't matter. What matters is that it's yours.

✓ **Self-check:** Your cheat sheet should be something you'd actually open the next time you sit down to write a prompt. If it looks generic, make it more specific to how you actually work.

What good looks like

After this homework, you should be able to write a four-part prompt for any task in under two minutes and get a useful first response more than 80% of the time. The other 20% is what iteration is for. That gap between "first draft" and "useful" closes with every prompt you write.

Lesson 06: How to use an API

Lesson 06 — How to use an API

From typing once to running a thousand times — while you make coffee.

AI for Dummies

Module 6

An API is how your *code* talks to AI instead of you. The model is the same — the door is programmable.

Why an API? — 1 vs 1,000

Chat

You type. You wait. You read the reply. You type again.
One exchange at a time.

To summarise 1,000 customer reviews you'd need to paste them one by one, every day, forever.

1 reply / minute

API

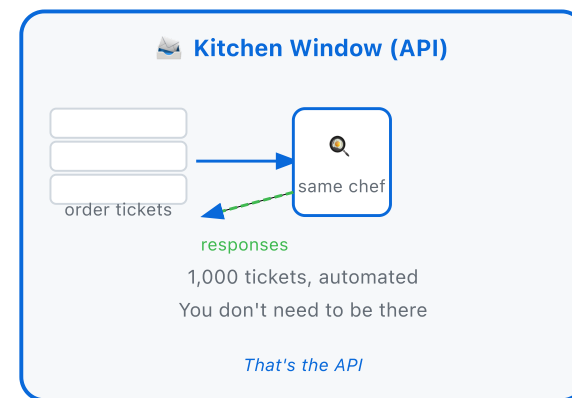
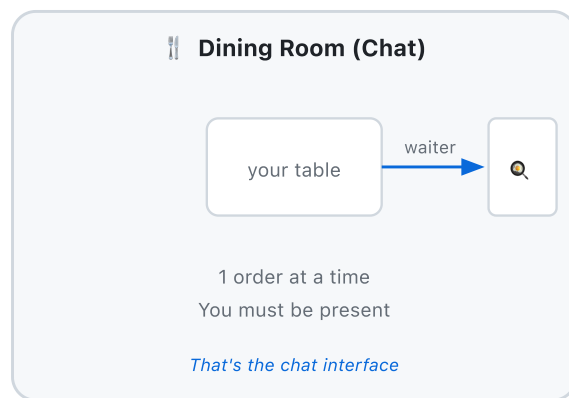
Your program sends prompts. It collects replies.
You're not even at the keyboard.

The same 1,000 reviews: a small Python loop, a coffee break, and a spreadsheet of results waiting for you.

1,000 replies / coffee break

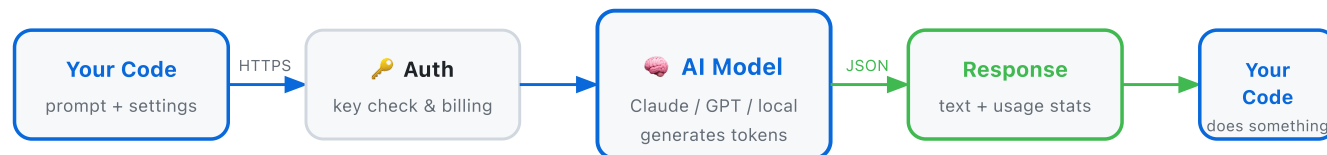
Rule of thumb: if you'd paste the same prompt more than a handful of times, the API is the right door.
Learn it once; use it everywhere.

The kitchen window — same chef, different door



Same restaurant. Same chef. Same food. Just a different, programmable doorway.

Anatomy of an API call



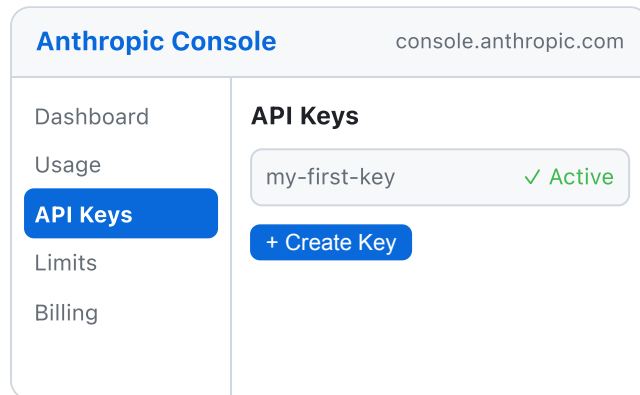
Round trip: typically 1–3 seconds for a short reply

What you send: model name, max tokens, your message list. That's it.

What comes back: the reply text, token counts, finish reason. Plain JSON.

Every language — curl, Python, JavaScript — wraps this same HTTP round-trip. The shape never changes.

Step 1 — Get an API key



1. Go to **console.anthropic.com**
2. Sign up or sign in
3. Click **API Keys** in the sidebar
4. Click **Create Key** — give it a name
5. Copy it *right now* — won't be shown again
6. Go to **Limits** → set a monthly cap

Homework tip: visit the Console, explore the Usage & Limits tabs, look around first — no key needed yet.

Your API key — treat it like a credit card number

⚠ **Leaked keys get used by strangers.** Your account gets the bill. Keys slip onto GitHub every day — people paste them into code they then commit. That's how cloud bills appear out of nowhere.

❌ Never do this

- Paste the key into a chat window
- Hard-code it in your Python file
- Put it in a file you'll commit to GitHub
- Share it in a forum post or email

✅ Always do this

- Store in an **environment variable**
- Read it with `os.environ` or the SDK auto-reads it
- Set a **spending limit** in the dashboard
- Rotate the key if you suspect a leak

Mac/Linux one-liner: `export ANTHROPIC_API_KEY=sk-ant-abc123` — add it to `~/.zshrc` to make it permanent.

Windows: System Properties → Environment Variables.

The simplest call — curl in the terminal

No library to install. `curl` is on every Mac and Linux machine already.

```
curl https://api.anthropic.com/v1/messages \      # ← the endpoint
-H "x-api-key: $ANTHROPIC_API_KEY" \           # ← key from env var, never in plain text
-H "anthropic-version: 2023-06-01" \          # ← API version header (required)
-H "content-type: application/json" \
-d '{
  "model": "claude-sonnet-4-6",                # ← which model to use
  "max_tokens": 1024,                          # ← ceiling on reply length
  "messages": [
    {"role": "user", "content": "Explain what an API is in one sentence."}
  ]
}'
```

What's the "-H"? HTTP headers — metadata that travels with the request. Like the envelope around a letter.

What's the "-d"? The data body — your actual prompt in JSON format. The model reads this.

Same call in Python — a complete AI program

Install once: `pip install anthropic`. Then this is a **complete working program**.

```
import anthropic

client = anthropic.Anthropic() # reads ANTHROPIC_API_KEY from environment automatically

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

print(message.content[0].text) # ← the model's reply, as a plain string
```

7 lines. That's it. The SDK handles all the HTTP, JSON, and authentication for you.

Add a loop around the `create()` call and you can process a spreadsheet of inputs while you sleep.

What just happened? — the round trip

▶ Replay animation

You send: model, max_tokens, your messages array — as JSON over HTTPS.

You receive: `message.content[0].text` — the reply text. Plus usage: how many tokens were used (for billing).

Total round-trip for a short reply: typically 1–3 seconds. For streaming responses you can receive tokens as they're generated — but that's a later lesson.

Local model — same shape, free & private

☁️ Cloud (Claude via Anthropic)

```
import anthropic

client = anthropic.Anthropic()

msg = client.messages.create(
    model="claude-sonnet-4-6",
    max_tokens=256,
    messages=[{"role": "user",
                "content": "Hello!"}],
)
print(msg.content[0].text)
```

🏠 Local (Ollama — free, offline)

```
from openai import OpenAI

client = OpenAI(
    base_url="http://localhost:11434/v1",
    api_key="ollama", # any string
)
resp = client.chat.completions.create(
    model="llama3.2",
    messages=[{"role": "user",
                "content": "Hello!"}],
)
print(resp.choices[0].message.content)
```

Notice: the structure is almost identical. Change the `base_url`, the `model` name, and the SDK — the rest of your code stays the same. *Learn the shape once.*

Ollama vs LM Studio — two local servers

Ollama

Runs in the background (terminal). Default port:

localhost:11434

```
base_url="http://localhost:11434/v1"  
model="llama3.2" # or gemma3, mistral...
```

LM Studio

GUI app — start the local server from the sidebar.

Default port:

localhost:1234

```
base_url="http://localhost:1234/v1"  
model="lmstudio-community/Meta-Llama-3..."
```

The only difference between the two: the port number. Everything else in your code is identical. Switching from Ollama to LM Studio is a one-character change.

Covered in depth in Lesson 04. Revisiting it here because it unlocks free, private API calls for any volume.

What is a token?

Cloud APIs don't charge per message or per word — they charge per **token**. Here's what that means in practice:

Rule of thumb: **1 token $\approx$ $\frac{3}{4}$ of a word $\approx$ 4 characters**

Text	Tokens	Why
hello	1	Common short word — single token
anti establish ment arian ism	6	Long rare word split into sub-word pieces
😊 (emoji)	1–3	Emojis are multi-byte UTF-8 sequences
The cat sat on a mat	6	6 words $\approx$ 6 tokens (short common words)

You pay for tokens **both ways**: what you send (input) *and* what the model generates (output). A typical short reply uses a few hundred tokens — a fraction of a cent. The cost is real when you loop over thousands of items.

Model pricing tiers — tiny to powerful

Prices are per million tokens. Same idea, very different scale.

⚡ Haiku (fast)		\$1 / \$5 per M
📊 Sonnet (balanced)		\$3 / \$15 per M
🧠 Opus (smartest)		\$15 / \$75 per M
🏠 Local model		\$0.00 always

Input / output rates. Check provider docs for current prices.

Start with Haiku for bulk, classification, or extraction tasks.
Upgrade to Sonnet only if quality falls short.

Set a spending limit in the Console *before* you run a loop. If your code has a bug and calls 10,000 times instead of 10, the cap stops the damage.

Set a spending limit — do this before anything else

💰 **The \$10 lesson:** a loop with a small bug once ran 50,000 API calls in 20 minutes. The monthly spending limit stopped it at \$10 instead of \$500. It takes 30 seconds to set. Set it now.

How to set it in the Anthropic Console:

1. Go to **console.anthropic.com**
2. Click **Limits** in the sidebar
3. Set **Monthly spend limit** — start with \$5 or \$10
4. Optional: set an **email alert** at 80% of the limit

Also limit at the code level:

```
# Always set max_tokens explicitly!  
# Without it, the model can return  
# thousands of tokens per call.  
message = client.messages.create(  
    model="claude-sonnet-4-6",  
    max_tokens=256,      # ← hard ceiling  
    messages=[...]  
)
```

Two safety layers, both required: Console spending limit (stops the bill) + `max_tokens` in code (limits each individual call's output length).

Local models — the cost is always \$0



Running Ollama or LM Studio?

The token count is irrelevant. Cost = \$0.

When to use local

- Processing **private or sensitive data** (medical, legal, personal)
- High-volume experiments where cloud cost would add up
- Offline work — no internet needed after model download
- Learning the API without any billing risk

The trade-off

- Quality is good but not at Opus level
- Needs **16 GB RAM** for comfortable use
- Initial model download: 4–8 GB
- Speed depends on your hardware (GPU helps a lot)

Smart path: prototype with the cloud (no setup), switch to local once cost or privacy is the bottleneck.
Because both use the same API format, the switch is one URL change.

Realistic example — classify a 1,000-row spreadsheet

This is the most common beginner use-case. The loop is the whole idea.

```
import anthropic, csv

client = anthropic.Anthropic()
results = []

with open("reviews.csv") as f:
    for row in csv.DictReader(f):          # ← iterate each row
        msg = client.messages.create(
            model="claude-haiku-4-5",      # ← cheapest model for bulk work
            max_tokens=20,                  # ← short reply: just "positive" / "negative"
            messages=[{"role": "user", "content":
                f"Classify as positive/negative: {row['review_text']}"
            }],
        )
        results.append({
```

1,000 rows at Haiku rate (~500 input + 20 output tokens each): roughly \$0.50–\$0.60 total. Less than a coffee.

Add a `time.sleep(0.1)` between calls if you hit rate limits. Or use the Batch API for large jobs — that's a later lesson.

Recap — what you now know

- An API lets your **code** call AI — automatically, at scale
- The shape: send messages → get reply → do something with it
- curl = terminal, no library. Python SDK = `pip install anthropic`
- Local models (Ollama / LM Studio) use the **same code shape**, cost \$0
- Tokens $\approx \frac{3}{4}$ word. You pay for both input and output
- API key = credit card number — store in env var, never in code
- Set a **spending limit** before you run any loop
- Start with **Haiku** for bulk work, upgrade only if needed
- `max_tokens` in every call — cap each reply's length
- Test with 1 call before looping over 1,000

 **Week 3 milestone:** touch the API. Run the Python example. Process a small batch of something real. That's the goal.

Theory page: live token counter, cost calculator, tabbed curl / Python / local code switcher — no install needed.

Lesson 07: real-world projects people actually build once they have a key — the pattern behind almost all of them.

How to use an API

Reading time: ~15 min · Prereqs: Lessons 01–05 · 3 interactive demos inside

Why this lesson exists

Everything so far has been about using AI as a human talking to a chat window. That's great — it's where most people start and where they spend most of their time. But there's a moment, usually around "I want to do this same thing to a hundred emails," or "I want to build a small tool for my team," when the chat window stops being the right doorway.

That moment is the API. This lesson explains what it is, how to get access to it, how much it costs, and shows you the exact code that makes it go — including the version that runs on your own machine for free.

1. What an API actually is

API stands for Application Programming Interface. The name sounds technical but the idea is simple: it's a way for your code to talk to another piece of software, just like you type into a chat window, but automatically and at scale.

Here's the analogy that makes it click: imagine a restaurant. The dining room is the chat interface — you sit down, order one thing at a time, a person brings it to you. Now imagine the kitchen has a small window in the wall where you can pass order tickets directly. You can pass one ticket, or a thousand tickets, and each one comes back with a completed dish. You can process them automatically. You don't need a waiter. *That's the API — the kitchen window.*

Same restaurant. Same chef. Same food. Just a different, programmable doorway.

For AI specifically, the API lets your program send a prompt and get back a response, exactly like the chat window does, but without you sitting there typing. You can run this in a loop, at any hour, as many times as you like.

2. Getting an API key

To use a cloud AI API, you need an **API key** — a long secret string of letters and numbers that identifies your account. Every call you make is billed to it. Think of it as your credit card number and your password combined into one string.

Treat your API key like a credit card number

Seriously. If someone gets your API key, they can make calls that bill *your* account. Keys get leaked on GitHub every day because people paste them directly into code. Follow these rules from day one:

- Never paste your key into a chat window, a forum post, or an email.
- Never put it directly in your code. Not even in a comment.
- Never commit a file that contains it to version control (GitHub, GitLab, etc.).
- Store it in an **environment variable** — a named setting on your computer that your code can read without the key ever appearing in the code itself.
- Set a **spending limit** in your provider's dashboard immediately after creating the key. If something goes wrong, the bill stops at that limit.

For Claude, you create a key at **console.anthropic.com**. Sign up or sign in, click *API Keys* in the sidebar, then *Create Key*. Copy it the moment it's shown — it won't be displayed again. Then go to *Limits* and set a monthly spending cap.

Other providers work the same way. OpenAI has *platform.openai.com*. Google Cloud has AI Studio. The steps are similar everywhere: create an account, create a key, store the key safely.

One thing that might surprise you: local models like Ollama don't need a real API key at all. They're running on your own machine, so there's no account to bill. You still pass an `api_key` parameter in the code (the SDKs require the field), but any string works — even just the word `"ollama"`.

3. Your first call

The simplest way to call an API is from the terminal, using a tool called `curl` that comes pre-installed on most computers. Here's what calling Claude looks like:

```
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."}
    ]
  }'
```

Notice `$ANTHROPIC_API_KEY`: that's reading the key from an environment variable. The key never appears in the command itself. You'd set this variable once on your machine (`export ANTHROPIC_API_KEY=sk-ant-...` on Mac/Linux) and then reuse it everywhere.

You send that request. You get a JSON response back containing the model's reply. That's the entire thing. Everything else — Python programs, web apps, automation scripts — is just this same idea wrapped in more convenient packaging.

4. The same thing in Python

Most people don't call APIs with curl for long. They write a small program, usually in Python, because then they can put the call inside a loop, store results in a file, and do something useful at scale. Install the Anthropic library once with `pip install anthropic`, and then:

```
import anthropic

client = anthropic.Anthropic() # reads key from environment automatically

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

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

That's a complete working AI program. Seven lines. Add a loop around it and a list of inputs and you can process a whole spreadsheet in the time it would take you to type the first message into the chat window.

5. Running a local model the same way

Here's one of the nicer things about Ollama and LM Studio (from Lesson 04): they both expose a local API server on your machine that follows the same format as OpenAI's API. So you can write code almost identical to the cloud version, and it runs entirely offline, at no cost, with no data leaving your computer.

With Ollama running in the background, the code looks like this:

```
from openai import OpenAI

client = OpenAI(
    base_url="http://localhost:11434/v1",    # your local Ollama server
    api_key="ollama",                      # any string works locally
)

resp = client.chat.completions.create(
    model="llama3.2",
    messages=[{"role": "user", "content": "Say hello in three languages."}],
)
print(resp.choices[0].message.content)
```

For LM Studio, the only change is the port number — its server runs on `1234` by default instead of `11434`. Everything else is identical.

This is worth pausing on. The code shape — create a client, call a method, get text back — is the same whether you're calling Claude in the cloud or a local Llama model on your laptop. The address changes, the model name changes. The shape doesn't. Learn it once and it works everywhere.

Want to see all three side by side with a copy button? Here they are:

[The three ways to call an AI API](#)


```
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 shape of the call is identical across all three — the only differences are the address you send to and the SDK you use.

6. Tokens and what things cost

Cloud AI APIs don't charge per message. They charge per **token** — a small chunk of text, roughly $\frac{3}{4}$ of a word. The tokenizer in the model reads your prompt one token at a time and generates a response one token at a time. You pay for both: the tokens *you send* and the tokens *the model sends back*.

This is billed per million tokens, so each individual call costs a tiny fraction of a cent. A short conversation might use a few hundred tokens — that's less than a thousandth of a dollar. The math changes fast when you run a loop over thousands of items, which is why knowing the price and setting a spending limit matters.

The $\frac{3}{4}$ -word rule

Roughly **1 token $\approx \frac{3}{4}$ of a word ≈ 4 characters**. Common words are usually 1 token. Rare or long words might be 2–3 tokens. Punctuation, spaces, and code symbols add tokens too. The widget below will show you how this scales with real text.

Token counter

Paste any text and see a live estimate. Or use the preset buttons to see how typical inputs scale.

Short email

Article paragraph

Long contract section

Paste any text here to count tokens...

0

WORDS

0

TOKENS (APPROX)

0

CHARACTERS

Roughly 1 token $\approx$ $\frac{3}{4}$ word $\approx$ 4 characters. AI bills in tokens both ways — your prompt AND its reply.

Notice how the "long contract section" preset might look like a few sentences but clocks in at several hundred tokens. Now imagine running that through an API loop for every contract in a law firm's archive. The token count — and the bill — adds up fast.

Two practical habits that prevent surprises:

1. **Start with the cheapest model that does the job.** The gap in capability between a \$1/M-token model and a \$15/M-token model is real, but it's often not relevant for simple tasks like classification, summarisation, or extraction. Use the heavy model when you need nuanced reasoning; use the light model for bulk work.
2. **Set a spending limit before you run anything in a loop.** Every major provider lets you set a monthly cap. If your code has a bug and calls the API ten thousand times instead of ten, the limit stops the damage.

7. How much does it actually cost?

Prices vary by provider and model. The widget below lets you calculate a realistic estimate for a specific use case. The "per 1000 calls" projection is especially useful

when you're thinking about automation at scale.

API cost calculator

Model

Claude Sonnet — \$3.00/M input · \$15.00/M output

Input tokens per call

1000

Output tokens per call

500

Number of calls for batch estimate

1000

Cost per call	\$0.0105
Cost for batch	\$10.50
Input tokens (batch total)	1,000,000 tokens
Output tokens (batch total)	500,000 tokens

Prices illustrative — check provider docs for current rates. Local model cost is \$0.00 regardless of volume; hardware electricity not included.

8. Cloud vs local — picking the right tool

Now that you've seen both options, here's a plain summary of when each makes sense:

- **Cloud API (Claude, OpenAI, Gemini):** use this when you need the best quality, when setup simplicity matters more than cost, or when you're on modest hardware. Pay-as-you-go, no GPU required. Data leaves your machine.
- **Local API (Ollama, LM Studio):** use this when privacy is non-negotiable (medical notes, legal documents, anything confidential), when you have high volume that would make cloud costs impractical, or when you want to experiment without any billing risk. Needs decent hardware — 16 GB RAM is comfortable for most useful models. Data stays put.

The practical path most people take: start with the cloud (fast, no setup), switch to local if cost or privacy becomes the issue. Because both speak the same API format, the switch is usually just changing one URL and one model name.

The environment variable trick

On Mac and Linux, open your terminal and run:

```
export ANTHROPIC_API_KEY=sk-ant-your-key-here
```

Now any program you run in that terminal session can call

```
os.environ.get("ANTHROPIC_API_KEY")
```

 in Python and get it without the key ever touching your code. To make it permanent, add that line to your `~/.zshrc` or `~/.bashrc` file. On Windows, use System Properties → Environment Variables.

9. The shape is the same everywhere

The biggest insight from this lesson is structural: however you call an AI API — curl, Python, JavaScript, a no-code tool like Zapier or Make — the shape of the interaction is always the same:

1. Send a message (your prompt, plus some settings like model and max tokens)
2. Get a reply (the model's response text, plus usage stats)
3. Do something with the reply (print it, save it, feed it into the next step)

Understand that shape and you can read any AI API code, in any language, and understand what it's doing. The vocabulary might change slightly (messages vs completions, content vs choices), but the structure doesn't. That's why learning one API transfers so directly to others.

Quick-reference: the five things to do before your first real call

1. Store your key in an environment variable (never in code)
2. Set a monthly spending limit in the provider dashboard
3. Choose the cheapest model that will realistically do the job

4. Set `max_tokens` so a runaway call can't return a novel
5. Test with a single call before looping over thousands of inputs

Up next

The **homework** for this lesson asks you to explore the Anthropic Console — look around, check the Usage tab, read the pricing page — without yet creating a key. Familiarity with the interface makes the next step (actually using it) much less intimidating.

Lesson 07 takes everything from this lesson and turns it into concrete examples: what people actually build once they have an API key, what beginner-friendly first projects look like, and the pattern behind almost all of them.

LAST-REVIEWED: 2026-06-04

Homework — How to use an API

Estimated time: ~30–40 min · No programming required · Free to complete

Three exploratory tasks. The goal this time is not to run any code — it's to become familiar with the environment you'd use before you write your first line. People who've looked around the Console, read the pricing page, and estimated a real cost rarely get surprised by their first bill. That's what we're building here.

Before you start

All three tasks are browser-only. You'll need a free account at console.anthropic.com for Tasks 1 and 2. Task 3 uses the [Theory page](#) you just read. No API key needed for any of these tasks — the goal is to explore and observe, not to run any calls.

Task 1 — Tour the Anthropic Console

The Console is where you'd manage your API keys, set spending limits, and check how much you've used. Before you generate a key, spend a few minutes just looking around so the interface isn't unfamiliar when you need it for real.

1. Go to console.anthropic.com and sign up or sign in. A free account is sufficient — you don't need to add a payment method to look around.
2. In the left sidebar, find and click **API Keys**. Notice the button to create a key — but *do not click it yet*. Just observe the interface. What information would you need to fill in? What would happen once you create a key?
3. Find the **Limits** or **Plans & Billing** section. Look for where you'd set a monthly spending cap. Note whether a limit is already set by default, and what the process would be to change it.
4. Find the **Usage** tab (sometimes called *Usage* or *Activity*). This is where you'd see a chart of how many tokens you've used over time. Even with no usage,

look at the chart axes: what units are shown? What time periods can you select?

5. In your notes, write down: (a) where you'd go to create a key, (b) where you'd go to set a spending limit, and (c) what the Usage tab shows. Three short sentences is fine.

✓ **Self-check: you can describe, without opening the Console again, where to find the API Keys page, the spending limit setting, and the Usage chart.**

Task 2 — Read the pricing page and estimate a real cost

This task makes the token-and-cost idea concrete by applying it to a real scenario. Use the Theory page's cost calculator alongside the official pricing page to check your estimate.

1. Go to anthropic.com/pricing (or search "Anthropic API pricing"). Find the table of model prices. Note the input and output price per million tokens for at least two models: one cheaper/faster one and one more capable one.
2. Now imagine this scenario: you want to use the API to summarise each customer review on a website. Each review is about 80 words. The summary should be about 40 words.
 - Estimate the input tokens for one review: use the $\frac{3}{4}$ -word rule from the Theory page.
 - Estimate the output tokens for one summary: same rule.
 - The website has **5,000 reviews**. How many total input tokens? How many total output tokens?
3. Open the [Theory page](#) and use the cost calculator widget to calculate the total cost for 5,000 calls with your estimated token counts. Try it with both the cheaper and the more capable model. Note both costs.
4. In your notes: write the two costs side by side. Is the cheaper model's result significantly cheaper? Would the quality difference matter for a summarisation task?
5. Finally, look at the official pricing page again and compare the numbers in the widget to the real published prices. Are they similar? (The Theory page uses

illustrative prices to show the concept — always check the live docs before making real decisions.)

✓ **Self-check: you have two cost estimates for the 5,000-review scenario — one per model — and you've compared the widget's illustrative prices to the current official ones.**

Task 3 — The token counter in the wild

This task builds intuition for token size, which is the thing most new API users underestimate. You'll use the token counter widget on the Theory page with three different real texts.

1. Open the [Theory page](#) and scroll to the **Token counter** widget.
2. Find a short email in your own inbox (or write a fictional one from memory — something like "Hi, following up on our meeting, can we schedule a call for Thursday?"). Paste it into the widget. Note the token count.
3. Now find a paragraph from an article, news story, or any web page. Paste it in. Note the count.
4. Finally, think of a real task you'd realistically want to automate with an AI: perhaps summarising a news article, classifying a support email, or extracting key dates from a document. Write a draft of the prompt you'd send — just a few sentences, not code. Paste it into the widget.
5. In your notes: for each of the three texts, write the word count and the token count. Then answer: for your proposed automation task, roughly how many tokens would the prompt use? If you ran it 1,000 times, how many total input tokens is that?

✓ **Self-check: you have three token-count observations from real text, and a back-of-the-envelope estimate for a 1,000-call batch of your own automation idea.**

Bonus (optional, ~15 min)

Explore the **Anthropic API documentation** at docs.anthropic.com. Don't try to read it all — just browse. Look at the *Messages* endpoint page and compare its JSON structure to the curl example in the Theory page. Find the list of available models and see if the names match what you've been reading about. Note one thing that surprised you or that you'd want to understand better. This kind of exploratory reading — getting comfortable in the docs before you need them — is a habit that distinguishes people who learn technical tools quickly from those who don't.

What "done" looks like

In your notes you should have:

- Three short sentences from Task 1: where to create a key, where to set a limit, what the Usage tab shows.
- Two cost estimates from Task 2: the 5,000-review summarisation scenario priced at a cheaper model and a more capable one, plus your observation on how the widget's prices compare to the official page.
- Three token-count observations from Task 3 and a rough token estimate for your own automation idea at 1,000 calls.

You've done something more useful than running code: you've walked through the environment, understood what things cost, and built an intuition for scale. The first time you run a real API call, you'll already know what to expect.

Lesson 07: Real-world examples

Lesson 07 — Real-world examples

Five categories. Prompts you can copy today — no coding required.

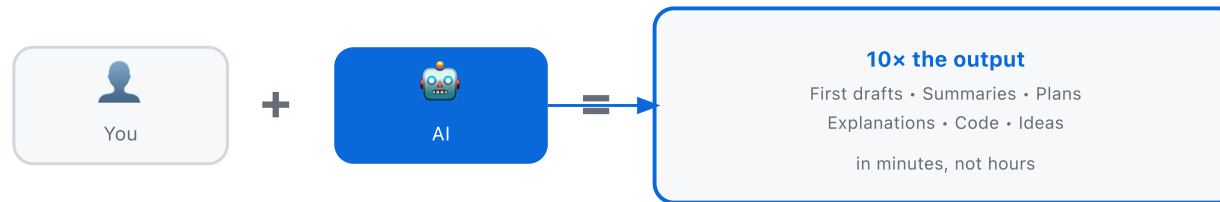
AI for Dummies

Module 7

This lesson is different. No theory — just **real prompts, real outputs, real transformations**. By the end you'll have a mental library you can reach for whenever you need it.

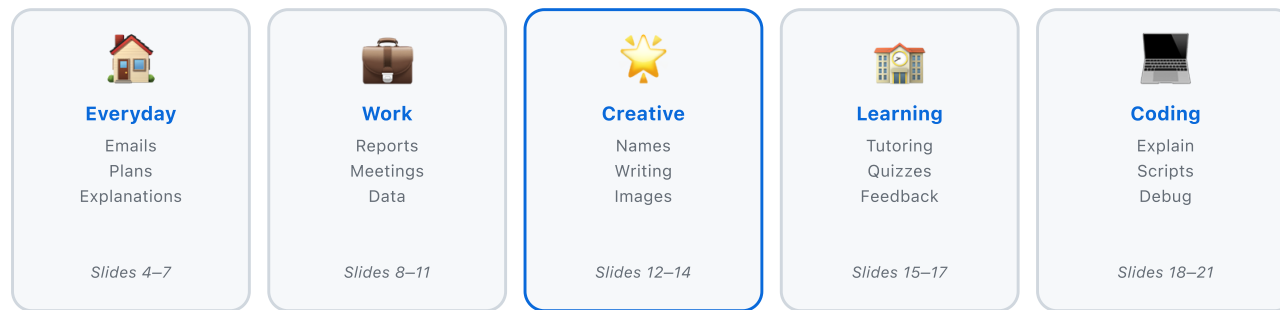
AI is a force multiplier

Same person. Same knowledge. Dramatically more output per hour.



The pattern across every example in this lesson: AI handles the first draft or the structured thinking — the hardest part to start. You steer, review, and decide.

Five categories — one lesson



Each section has a real prompt box you can copy. No theory — just examples.



Category 1 — Everyday life

The most underestimated category. These are the tasks we do every day — for family, home, and personal life — that consume real time and mental energy. AI handles all of them fast.

What belongs here:

- Writing emails you've been avoiding
- Summarising documents you don't want to read
- Planning trips, meals, workouts with constraints
- Understanding confusing things (contracts, medical reports, tax forms)

The pattern:

Take this *[messy input]*
and give me *[clean output]*
for *[my situation]*.

Messy in → polished out. That's the whole idea.

Everyday #1 — messy thoughts → polished email

BEFORE — YOUR RAW NOTES

"need to tell landlord roof is leaking again, last time took 3 weeks, really frustrated, want it fixed by friday but don't want to sound rude, they ignored my last message"

AFTER — AI OUTPUT

Subject: Roof leak — second notice

Hi David,

I'm writing again about the roof leak I reported three weeks ago. Unfortunately it persists and is getting worse. Could you confirm a repair date by Friday? I'd appreciate a quick response this time.

The prompt that did it:

Turn my notes into a polite but firm email to my landlord. Under 80 words. Flag the history (previous ignored message). Notes: [paste above]

Why this works: you gave the tone instruction (*polite but firm*), the constraint (*under 80 words*), and the context (*previous message ignored*). The AI threaded all three.

Thanks,
Maria

Everyday #2 — read a 12-page contract in 30 seconds

THE TASK

A 12-page rental agreement you received on a Friday evening. Dense legalese. You're moving in Monday.

AI OUTPUT — 5 THINGS TO KNOW

1. Notice period is **60 days** (not the standard 30).
2. Pets require written approval — not just verbal.
3. Landlord can enter with **24 hr notice**.
4. Deposit is non-refundable if lease breaks before 6 months.
5. ⚠️ Unusual clause: you pay for any plumbing repairs under €200.

The prompt:

Here is a rental contract. Give me the 5 most important things I need to know as a tenant, in plain English. Flag anything unusual or that I should be concerned about.

[paste document text here]

Caveat: use this to know *what to read carefully* in the original — not as a replacement for reading the key clauses yourself.

Everyday #3 — plan a trip around your real constraints

The prompt:

Plan a 5-day trip to Lisbon for 2 adults in late October. Budget: ~€1 500 total excluding flights. We like food, walking, and history. We hate tourist traps. Give a day-by-day outline with 2–3 activities per day and a rough daily cost.

SAMPLE AI OUTPUT (DAY 1)

Day 1 — Alfama & Fado

Morning: walk Alfama, skip the main square crowds — head to Miradouro da Graça instead. Afternoon: MAAT museum (€5). Evening: dinner at a neighbourhood tasca in Mouraria, followed by a live fado show (~€35 pp). *Est. day cost: €80 for two.*

Works the same way for:

- Weekly meal plan (with dietary constraints)
- Workout schedule (with equipment limits)
- Budget for a home renovation
- Study plan for an exam

Constraints make better prompts. €1 500 total, hate tourist

traps is ten times more useful than "plan a Lisbon trip."

Everyday #4 — same topic, two levels

Ask for the simple version, then the expert version. Same AI, same conversation.

SIMPLE VERSION

Explain compound interest like I'm 10 years old, with a simple analogy.

"Imagine you put €100 in a magic piggy bank. Every year it adds 10% of whatever is inside — not just the original €100. So after year 1 you have €110, year 2 you have €121, and it snowballs. Money making more money."

EXPERT VERSION (FOLLOW-UP)

Now explain it for a finance professional. Give the formula and a worked example with monthly compounding.

$A = P(1 + r/n)^{nt}$. At $P=€10\,000$, $r=6\%$, $n=12$, $t=10$:
 $A = €10\,000 \times (1.005)^{120} = \textbf{€18\,194}$. The monthly compounding adds ~€600 vs annual compounding over the decade.

Try this pattern on anything confusing — a news story, a contract clause, a medical term, a physics concept. Same question, two very different explanations, back to back.



Category 2 — Work

Knowledge workers spend a large fraction of their day on writing, meetings, and communication. These are not the core of most jobs — they're the scaffolding around it. AI handles scaffolding well.

High-value work prompts:

- Meeting notes → action items + owners
- Bullet points → executive report
- Data table → trend analysis
- Technical text → customer-facing copy

The prompt pattern:

Here is *[raw material]*.
Turn it into *[structured output]*
for *[audience]* in *[tone/format]*.

Work #1 — meeting notes → action list

BEFORE — RAW NOTES (45-MIN MEETING)

"discussed new landing page - tom said he'd get designs by eow. budget still unclear. marketing wants analytics set up before launch. dev says 2 weeks if designs on time. need to confirm launch date maybe 15th? sarah to check legal about disclaimer. follow up on email list migration - was supposed to be done last week."

AFTER — STRUCTURED OUTPUT

Action items:

- Tom — Landing page designs by **Fri EOW**
- Sarah — Legal check on disclaimer copy (*no deadline set*)
- Dev team — 2-week build once designs received
- [Unassigned] — Confirm launch date (~15th)
- [Unassigned] — Email list migration (*overdue!*)

Open questions: Budget unconfirmed. Analytics owner?

Here are my raw meeting notes. Extract all action items with owner + deadline (or "unassigned"). Also list open questions. Notes: [paste]

Work #2 — draft a report from notes

The prompt:

Write a one-page Q1 sales summary report. Key facts: revenue €420k (up 12% vs Q4), top product was the Pro plan, biggest challenge was supply chain delays in March. Audience: executive team. Tone: professional, direct. Include a 3-bullet highlights section at the top.

MOCK AI OUTPUT

Q1 2026 Sales Summary

Highlights:

- Revenue: **€420k** — up 12% quarter-over-quarter
- Pro plan drove 58% of new revenue
- March supply delays cost est. €30k in deferred deals

Performance: Q1 exceeded Q4 across all key metrics. The Pro plan outperformed projections by 18%...

Risk: Supply chain vulnerability remains unresolved. Recommend a contingency review before Q2.

The blank page problem disappears. Give AI the skeleton; you add the company-specific numbers, context, and political nuance. Takes 5 minutes, not 2 hours.

Work #3 — ask questions about a spreadsheet

The prompt:

Here is a table of monthly sales by product for the past 12 months. What are the three biggest trends? Which product is growing fastest? Is anything declining? [paste table]

What a pasted table might look like, and what AI extracts:

Month	Pro	Starter	Enterprise
Jan	€38k	€21k	€15k
Feb	€42k	€20k	€17k
Mar	€45k	€18k	€19k
Apr	€51k	€17k	€22k

AI analysis:

- Pro: **+34% over 4 months**, fastest grower
- Enterprise: **+47% growth**, accelerating
- Starter: **-19% trend**, declining — investigate

Work #4 — adapt content for a different audience

TECHNICAL ORIGINAL

"The new webhook integration enables real-time event-driven architecture by pushing POST requests to a configured endpoint whenever state transitions occur. Supports HMAC-SHA256 signature verification. Retry logic uses exponential backoff up to 5 attempts."

Rewrite this for our non-technical sales team. No jargon. Focus on what the customer can do and why it helps them. Under 60 words. Friendly tone.
Original: [paste]

CUSTOMER-FACING OUTPUT

New: instant updates to your other tools

Connect your account so your CRM, Slack, or spreadsheet updates the moment something changes — no more manual exports or waiting overnight. Fast, secure, and if your system is briefly offline, we automatically retry for you.

Also works in reverse: translate customer feedback into a technical bug report. Or adapt one document for five different audiences in minutes.

Category 3 — Creative & brainstorming

AI doesn't replace your creativity. It **unsticks you** when you have nothing, and **gives you options** when you have too many possibilities to start. Your taste is still the filter.

The prompt: give me 10 names for a dog-walking app. Short, memorable, playful. Avoid "paw" or "wag" — taken. Mix single words and short phrases.

Barkly

Walkr

Fetch & Go

Sniff Club

Roamio

TailTime

GoodBoy

Strut

LeashUp

RunRover

The trick: your taste steers the process. "I like #1 and #5. Give me 10 more in that direction." You're not accepting output — you're curating it.

Creative — getting unstuck on writing

Stuck on a paragraph? Ask for three different versions to react to. Pick one. Polish it.

The prompt:

I'm writing a travel blog about Kyoto in autumn. Stuck on the opening paragraph. Give me three very different options: one lyrical and atmospheric, one punchy/direct, one that opens with a specific anecdote. I'll choose the one that fits my voice.

LYRICAL OPTION

"The maple leaves come down in Kyoto like a slow red argument with gravity — each one holding on a moment longer than it should before letting go."

PUNCHY OPTION

"Everyone tells you Kyoto is beautiful in autumn. They're not wrong. But they don't tell you how crowded it is at the famous spots, or where to go instead."

Key mindset: you often discard all three — but seeing the contrast tells you what you *actually* want to write. That's the value.

Creative — generating images with text

Tools for this:

- **Midjourney** — highest quality, Discord-based
- **DALL•E 3** — inside ChatGPT, easy to use
- **Adobe Firefly** — good for commercial use
- **Stable Diffusion** — free, runs locally

Good for: blog illustrations, social post images, presentation visuals, logo brainstorm, product mockups, poster ideas.

Example image prompt:

A flat-style illustration of a cozy home office at night, warm lamp light, plants on the desk, minimal and modern. Muted color palette. No people.

Prompting images follows the same rules as text: be specific about style, mood, and what to exclude. "Flat-style, muted palette, no people" is a complete brief.



Category 4 — Learning: the 2 a.m. tutor

A patient tutor for any subject, available at any hour, that adapts to your exact level and never makes you feel stupid for asking again. This is the most powerful use case for serious learners.

The setup prompt:

You are a patient tutor helping me learn basic economics. I'm a complete beginner. Start with supply and demand — explain it simply with a real-world example, check that I understand, and only move on when I say I'm ready.

What you get:

- A plain-English explanation at your level
- A check-in: "does that make sense?"
- Deeper content when you ask for it
- Unlimited "try a different analogy" rounds
- Moves at your pace, not a class schedule

Human tutors run out of analogies. AI doesn't.

Learning — quiz yourself on anything

Reading is passive. Active recall — being tested — is how things stick. Research-backed, easy to set up.

The prompt:

```
Ask me 10 multiple-choice questions on the French Revolution. Give me one question at a time. After I answer, tell me if I'm right and explain briefly why. Keep a running score. Go.
```

Sample Q&A exchange:

AI: Q1/10 — What year did the French Revolution begin?

A) 1776 B) 1789 C) 1799 D) 1804

You: B

AI: Correct! 1789. The storming of the Bastille on July 14th is the symbolic start. Score: 1/1. Next question...

Works for any subject: history, science, coding languages, foreign vocabulary, material from a course you're enrolled in. You can even paste in the content: "Based on this chapter, create 8 short-answer questions."

Learning — feedback on your writing

The prompt:

Here is my essay draft. Give me honest, structured feedback on three things: 1) strength of the argument, 2) clarity of the structure, 3) writing quality. Don't rewrite it — just tell me where it's strong and where it needs work.

[paste essay]

AI feedback example:

Argument (7/10): Your thesis is clear, but paragraph 3 contradicts the claim in paragraph 1 without acknowledging the tension.

Structure (6/10): The conclusion introduces a new idea that should appear earlier. Consider moving it to the second section.

Writing (8/10): Clear and readable. Watch out for passive voice in the opening — it weakens the authority of your first impression.

This kind of structured feedback — normally requiring a mentor or writing group — is now available any time for free. Same works for code, business plans, or any reasoning exercise.



Category 5 — Coding (even for near-beginners)

You do *not* have to be a developer. The best coding use cases are often for people who know a *little* code — enough to have a problem, not enough to solve it alone.

What near-beginners use it for:

- Explain what a piece of code does, line by line
- Write small scripts for repetitive tasks
- Debug: paste an error, get the fix + explanation
- Learn: “explain what a for-loop does, 3 examples”

The ask-for-explanation habit:

Always add “explain each line” or “explain why” to any coding prompt. You get the script *and* you understand it. That’s how near-beginners actually level up.

Coding #1 — write me a script

The prompt:

Write a Python script that goes through all files in a folder and renames any .jpeg file to .jpg. I'm on Mac. Keep it simple and explain each line.

AI OUTPUT

```
import os

folder = "/Users/you/photos"  # ← change this

for filename in os.listdir(folder):
    if filename.endswith(".jpeg"):
        old = os.path.join(folder, filename)
        new = old[:-5] + ".jpg"
        os.rename(old, new)
        print(f"Renamed: {filename}")
```

After you get this, ask follow-ups: "What if I want it to include subfolders?" or "Can it ask for confirmation first?" The conversation continues until it does exactly what you need.

Coding #2 — explain this code line by line

Inherited mystery code:

```
def process(data):  
    result = {}  
    for item in data:  
        k = item['id']  
        result[k] = result.get(k, 0) + 1  
    return {k: v for k, v  
            in result.items()  
            if v > 1}
```

Explain what this code does. Go line by line for anything complicated. Assume I know Python basics but not advanced concepts.

AI EXPLANATION

What it does overall: finds duplicates. It counts how many times each `id` appears in the list, then returns only the ones that appear more than once.

Line by line:

- `result = {}` — empty dictionary to store counts
- `result.get(k, 0) + 1` — if key exists add 1, otherwise start at 0
- Last 3 lines — dictionary comprehension: keep only entries where count > 1

Coding #3 — find and fix the bug (with the WHY)

The prompt:

I got this error running my script. Tell me what caused it and how to fix it. Explain why it broke so I understand the concept.

Error:

`TypeError: can only concatenate str (not "int") to str`

Code:

```
age = 25
print("I am " + age + " years old")
```

AI RESPONSE

What caused it: Python won't let you join a string and an integer with `+`. `"I am "` is a string; `25` is a number. They're different types.

The fix — two options:

```
# Option 1: convert int to string
print("I am " + str(age) + " years old")

# Option 2: f-string (cleaner)
print(f"I am {age} years old")
```

The “explain why it broke” instruction is the key. You understand the concept for next time — not just copy-pasted code you don't understand.

The pattern across all five categories

AI is a force multiplier for getting started. In every single example in this lesson, the AI did one of two things: it handled the *first draft*, or it provided the *structured thinking* that was blocking you. You steered, reviewed, and decided.

What AI is great at:

- First drafts of anything written
- Transforming messy input into clean output
- Generating options for you to react to
- Explaining things at the right level
- Writing small code tasks from a description

What you still do:

- Provide the real context and constraints
- Verify anything consequential
- Make the final call
- Polish what doesn't sound like you
- Know when the output is wrong

Honest limitations — where AI falls short

Use with caution:

- **Real-time facts** — AI doesn't know today's news or live prices. Always verify.
- **Precise arithmetic** — AI can get numbers wrong. Double-check calculations.
- **Niche expertise** — obscure legal, medical, or technical edge cases. Starting point, not final authority.

The simple rule:

If being wrong has real consequences — health, money, legal, sending to customers — treat AI as a **research assistant**, not a final authority. Use it to get oriented fast, then verify what matters.

The people who get the most value from AI are the ones who *also* know when not to trust it.

Use it on something real this week

Everything in this lesson becomes real skill the moment you use it on an **actual task from your life**. Not a practice exercise — a real email you've been avoiding, a document in your inbox, a piece of writing you're stuck on.

Pick one this week:

- Draft an email you've been putting off
- Summarise a document sitting in your inbox
- Plan a trip, meal, or project with constraints
- Get help preparing for a meeting

Or:

- Ask for an explanation of something confusing
- Quiz yourself on something you're learning
- Name a project or brainstorm a list
- Paste an error message and ask for the fix

Theory becomes skill at the moment of first use on something real.

What's coming next

Lesson 08 — AI in your own projects

From scripts to small apps. Cloud vs local. How to fold AI into something you're building — even if you're not a developer.

Lesson 09 — Safety, privacy & cost

What AI can't do. Where it goes wrong. How to stay protected and in control.

Before you go:

Read the Theory page for the full interactive use-case explorer (30+ prompts), the before/after gallery, and the pattern recognizer quiz. Each prompt is copy-ready.

The five prompt patterns in this lesson — transform / structure / generate-options / tutor-dialogue / write-and-explain — cover 90% of what most people will ever need.

Real-world examples

Reading time: ~15 min · Prereqs: Lessons 01–06 · 3 interactive demos inside

Why concrete examples beat abstract advice

You've learned what AI is, how to access it, and how to prompt it well. Now the question everyone eventually asks is: *"But what do I actually use it for?"*

This lesson is the answer. Not a list of categories, but real, specific, copy-and-use examples across five domains: everyday life, work, creative, learning, and coding. By the end, you'll have a mental library of prompts you can reach for whenever you need them.

The common thread across all five categories is the same: **AI is a force multiplier for tasks you could do yourself, but would rather do faster — especially first drafts and "getting started."** Understanding this frame prevents both over-reliance (treating AI as an oracle) and under-use (never trying it because you're not sure if it "counts" as a valid use case).

How to use this page

Read through once to build the mental map. Then use the interactive Explorer below as a personal prompt library you can return to whenever you need a starting point for a new task. The Before/After gallery shows you what the difference between raw notes and polished AI output looks like in practice. The Pattern Quiz at the end tests whether you've internalized the categories.

Category 1 — Everyday life

The most underestimated use case. The tasks we do every day — writing emails, reading documents, making plans, understanding confusing things — consume real

time and mental energy. None of them require AI to be "smart" in any deep sense. They require drafting speed, patience, and the ability to explain things clearly. AI is excellent at all three.

Turning messy thoughts into clean emails

Most people write emails in three stages: first, they get the thoughts out in a messy, informal way; then they rewrite it into something readable; then they second-guess the tone and rewrite again. AI collapses this into one step. You dump your thoughts (in any order, in any style), describe who you're writing to and what you want them to feel, and ask for a draft. You read it, adjust anything that doesn't sound like you, and send.

This works for difficult emails especially well. Chasing an overdue invoice, responding to a complaint, telling a colleague something awkward — the hard part of these emails isn't the writing, it's navigating the tone. You can tell AI "be firm but not aggressive" or "be warm but not apologetic" and it will thread that needle better than most people do under pressure.

Reading without reading

Rental agreements, insurance policies, terms of service, long articles — everyone has a pile of documents they know they should read but haven't. Paste the text into an AI chat (or a big chunk of it) and ask: "What are the five things I need to know? Flag anything unusual or that I should be concerned about." You'll get a genuine summary in under a minute.

Important caveat: for anything legally or financially consequential, the AI summary gets you oriented, not off the hook. Use it to know *what to read carefully* in the original document, not as a replacement for reading it.

Explaining things at exactly the level you need

AI is uniquely patient about re-explaining things in different ways. You can ask for an explanation "like I'm 10 years old," then follow up with "now go deeper" or "explain the technical side." You can ask it to use a specific analogy ("explain this the way you'd explain it to someone who knows cooking but not physics"). This kind of adaptive explanation is something even excellent teachers struggle to do consistently under time pressure — AI does it every time, at no extra cost.

Category 2 — Work

Knowledge workers spend a large fraction of their day on writing, meetings, and communication. These activities are not the core of most jobs — they're the scaffolding around it. AI handles scaffolding well. Here are the most valuable work use cases.

From meeting notes to action items

Every meeting produces some notes. Often they're patchy, unformatted, and scattered across someone's laptop. Paste them into an AI and ask: "Extract the action items. For each, note who owns it and the deadline. If an owner or deadline isn't mentioned, flag it as unassigned." What comes back is a structured task list you can copy into your project management tool immediately.

You can also ask: "What decisions were made?" and "What questions were left open?" These three prompts together turn raw notes into a complete meeting record in about 90 seconds.

Drafting reports, proposals, and job descriptions

The blank page is the hardest part of any writing task. With AI, you never face it. Describe what you need (the document type, the audience, the tone, the key points to cover) and ask for a first draft. The draft will almost certainly need editing — it won't know your company's style, your specific numbers, or the political context of the proposal. But it gives you a structured starting point to react to, which is much faster than starting from scratch.

This is especially useful for document types you rarely write. If you have to write a job description for the first time, or a board-level summary, or a vendor RFP, you may not know the genre conventions. AI does. It will produce a structurally sound template that you fill in with real content.

Analysing data in plain English

You don't have to be a data analyst to ask intelligent questions about a dataset. Paste a table of numbers (copied from a spreadsheet) and ask: "What are the three biggest trends in this data? Which category is growing fastest? Is anything declining?" AI will interpret the numbers and surface patterns that would take a non-analyst much longer to find manually.

This works best with small-to-medium tables (up to a few hundred rows). For larger datasets, you'll need a code-based approach, but for everyday business data — sales figures, survey results, team metrics — conversational data analysis is remarkably effective.

Category 3 — Creative

AI's creative applications get misunderstood in two opposite ways: some people think it's too good ("it's replacing creatives") and others think it's useless ("the output has no soul"). The truth is it's neither. AI is an excellent *collaborator* for creative work, specifically for the phases of the process where you need quantity before quality.

Brainstorming at scale

Good brainstorming requires generating a lot of options without judging too early. Most people are bad at this alone — they generate three ideas, get stuck, and pick the least-bad one. With AI, you can generate 15, 20, or 30 options in a minute. Your job becomes curation, not generation.

The trick is to give constraints. Not "give me name ideas" but "give me 15 name ideas for a dog-walking app, short and memorable, playful tone, avoid anything with 'paw' or 'wag' since those are taken, mix of single words and two-word phrases." Tight constraints produce better brainstormers. If you like #4 and #11, say so and ask for 10 more in that direction.

Getting unstuck on writing

Every writer knows the feeling: you need a paragraph or a page, you know roughly what it should say, but you can't find the words. AI is genuinely useful here — not to write the thing for you, but to give you options to react to. Ask for three very different versions (lyrical, direct, anecdote-led) and see which one sparks something. You'll often end up discarding all three but knowing clearly what you actually want to write. That's the value: the contrast, not the output.

Editing and tone-shifting

You wrote something. It's okay, but it's too long, or too formal, or too casual. AI is a patient editor. "Make this punchier, cut it by 30%" or "rewrite this in a warmer, more conversational tone" or "make this suitable for a technical audience who already know

the domain" — these instructions produce reliable, fast results. You can chain edits: take the AI's version, ask for another pass, and so on until it's close to what you want.

Category 4 — Learning

This is perhaps the most powerful use case for anyone who is serious about learning a new skill or subject. AI removes the two biggest barriers to self-directed learning: access to a patient expert, and the ability to practice interactively.

The adaptive tutor

"You are a patient tutor helping me learn basic economics. I'm a complete beginner. Start with supply and demand — explain it simply, check that I understand, then move on when I say I'm ready." This prompt sets up an interaction that continues at your pace, adapts to what you already know, and never loses patience. You can stay on supply and demand for as long as you need before moving on. No textbook does this.

The key is to push back when something doesn't land: "I still don't understand the bit about equilibrium price — can you try a different analogy?" A human tutor might run out of ways to explain something. AI has essentially unlimited alternative explanations.

Active recall through quizzing

Reading is passive. Being tested is active. Decades of educational psychology research confirm that active recall — trying to retrieve information from memory — is one of the most effective ways to learn. AI makes it trivially easy to quiz yourself on anything. "Ask me 10 multiple-choice questions on the French Revolution. Give me one at a time. After I answer, tell me if I'm right and explain why. Keep track of my score."

This works for any subject: history, science, a programming language, a foreign language, or material from a course you're enrolled in. You can even paste in the content you want to be quizzed on: "Based on this chapter summary, create 8 short-answer questions for me to practice."

Getting feedback on your work

Writing an essay, drafting a business plan, reasoning through a problem — you can get structured feedback from AI on any of these. "Here is my essay draft. Give me honest feedback on the argument, the structure, and the clarity. Don't rewrite it — just tell me

where it's strong and where it's weak." This kind of feedback loop, which normally requires access to a mentor or writing group, is available any time for free.

Category 5 — Coding (even for near-beginners)

You do not have to be a developer to benefit from AI coding help. Some of the most valuable use cases are for people who know a little code — enough to understand what a script does and modify it, but not enough to write it from scratch. Here's what that looks like in practice.

Writing small scripts for repetitive tasks

The sweet spot for near-beginner coding with AI is small, well-defined tasks: rename all the files in a folder, combine several CSV files into one, send a formatted email from a spreadsheet, pull data from a website into a table. These tasks take an experienced developer about 20 minutes to write. They take a non-developer potentially never, because the blank page of a code file is impenetrable without a starting point.

With AI: describe the task in plain English, specify your operating system and any constraints, and ask it to explain each line. You get a working script and you understand what it does. If it doesn't work the first time (this happens), paste the error message back in and ask "what's wrong?"

Explaining code you didn't write

You inherited a script. Or you found something on the internet that almost does what you need. Or you wrote something yourself six months ago and can't remember how it works. Paste it into AI and ask: "Explain what this code does. Go line by line for anything complicated. Assume I know Python basics but not advanced concepts." What comes back is a plain-English walkthrough of every significant part of the code.

Debugging

Error messages are designed for compilers, not humans. They're often cryptic, referencing line numbers and jargon that means nothing without context. AI translates them. Paste the error message, paste the relevant code, and ask: "What's causing this error and how do I fix it?" You'll get both the fix and an explanation of *why* it was broken — which means you'll understand the concept for next time.

INTERACTIVE 1 OF 3 — USE-CASE EXPLORER

Browse real prompts by category. Click any "Copy this prompt" button to paste it directly into an AI chat.



Everyday



Work



Creative



Learning



Coding

Email from bullet points

PROMPT TYPE: DRAFT FROM NOTES

Turn my notes into a polite but firm email to my landlord. Keep it under 120 words. My notes: roof still leaking after 3 weeks, last repair didn't hold, want it fixed by Friday, don't want to sound aggressive but need to be clear.



Copy this prompt

Document summary

PROMPT TYPE: SUMMARISE LONG TEXT

Here is a rental contract. Give me the 5 most important things I need to know as a tenant, in plain English. Flag anything unusual or that I should be concerned about. [paste document]



Copy this prompt

Explain it simply

PROMPT TYPE: ADAPTIVE EXPLANATION

Explain compound interest like I'm 10 years old, using a simple analogy. Then give me a quick worked example with real numbers.



Copy this prompt

Trip planner

PROMPT TYPE: PLAN WITH CONSTRAINTS

Plan a 5-day trip to Lisbon for 2 adults in late October. Budget: around €1500 total excluding flights. We like food, walking, and history. We hate tourist traps. Give a day-by-day outline with 2-3 activities per day.



Copy this prompt

Meal plan for the week

PROMPT TYPE: PLANNING WITH CONSTRAINTS

Difficult conversation script

PROMPT TYPE: PREPARE FOR HARD TALK

Create a 5-day meal plan for 2 people. Constraints: one person is lactose intolerant, we have 30 minutes max on weeknights, budget is €60 for the week's groceries. Include a shopping list.

 Copy this prompt

Help me prepare for a difficult conversation. I need to tell a friend that their constant lateness is affecting our friendship. I want to be honest but not hurtful. Give me an opening line and 3 key points to make clearly.

 Copy this prompt

All prompts are generic starting points. Replace the bracketed placeholders with your real content before pasting into an AI chat.

Why the "before" matters as much as the "after"

One thing the prompts above share: they all start with something real that you already have. Meeting notes, raw thoughts, a block of code, a chapter of a textbook. This is intentional. AI doesn't create from nothing — it transforms what you give it. The better and more specific your input, the more useful the output.

The before/after gallery below shows exactly what this looks like in practice: messy raw input goes in, and polished, usable output comes out. More importantly, it shows the prompt that made the transformation happen.

INTERACTIVE 2 OF 3 — BEFORE / AFTER GALLERY

Three real transformations. Click "Show prompt used" to see exactly what was sent to the AI.

EXAMPLE 1 — MESSY NOTES → PROFESSIONAL EMAIL

BEFORE (RAW NOTES)

need to tell my manager i cant make the monday meeting. dentist appt i forgot to put in calendar. also need to ask if we can move it to tuesday afternoon or i can just get the notes. dont want to seem unprofessional. we

AFTER (POLISHED EMAIL)

Subject: Monday meeting — scheduling conflict

Hi Sarah,

I have a dentist appointment on Monday that conflicts with our 10 a.m.

have a big presentation thursday so this is kind of important timing.

meeting — I apologise for the late notice.

Would it be possible to move the meeting to Tuesday afternoon? If that doesn't work for the team, I'm happy to catch up from the notes instead. With the Thursday presentation coming up, I want to make sure I don't miss anything important.

Let me know what works best.

Best,
Alex

 Show prompt used

EXAMPLE 2 — RAW MEETING NOTES → ACTION LIST

BEFORE (RAW MEETING NOTES)

discussed new landing page - tom said he'd get designs by eow. budget still unclear. marketing wants analytics set up before launch. dev team said 2 weeks for build if designs come on time. need to confirm launch date - maybe 15th? sarah to check with legal about the disclaimer copy. follow up on the email list migration - that was supposed to be done last week.

 Show prompt used

AFTER (STRUCTURED ACTION LIST)

Action items

-  Tom — Landing page designs by end of week
-  Sarah — Check with legal on disclaimer copy (no deadline set)
-  Dev team — 2-week build once designs received
-  [Unassigned] — Confirm launch date (possibly 15th)
-  [Unassigned] — Follow up on email list migration (overdue)

Open questions: Budget not yet confirmed. Analytics setup needed before launch — who owns this?

EXAMPLE 3 — TECHNICAL EXPLANATION → PLAIN ENGLISH FOR CUSTOMERS

BEFORE (INTERNAL TECHNICAL DOC)

The new webhook integration enables real-time event-driven architecture by pushing POST requests to a configured endpoint whenever state transitions occur in the pipeline. Supports HMAC-SHA256 signature verification for security. Retry logic uses exponential backoff up to 5 attempts. Max payload size is 256 KB. Latency is typically under 200ms.

AFTER (CUSTOMER-FACING SUMMARY)

What's new: instant updates to your systems

You can now connect Basil to your own tools so they update automatically the moment something changes — no more manual exports or waiting for a daily sync.

It's fast (usually under a second), secure, and reliable. If your system is temporarily offline, Basil will keep retrying automatically for you.

Ask your developer to set it up — it takes about 15 minutes.

 Show prompt used

Notice that in every case, the raw input was imperfect — and that was fine. AI works with what you give it.

Recognising which category a task belongs to

After working through enough examples, you start to develop a mental reflex: you see a task and immediately know the shape of the prompt that will handle it. Before that reflex kicks in naturally, it helps to be deliberate about categorisation.

Each of the five categories has a characteristic "prompt shape":

- **Everyday life** tasks tend to involve transforming something messy into something polished, or explaining something to yourself. Shape: "take this [messy input] and give me [clean output]."
- **Work** tasks tend to involve structuring information, drafting documents, or extracting insights. Shape: "here is raw material, turn it into [structured output] for [audience]."

- **Creative** tasks tend to involve generating options to react to. Shape: "give me N options for [creative thing], I'll choose and iterate."
- **Learning** tasks tend to involve dialogue: explanation, testing, feedback. Shape: "explain [thing] to me at [level], then check my understanding."
- **Coding** tasks tend to involve writing, explaining, or fixing code with a clear success condition. Shape: "write/explain/fix [code thing], explain why."

INTERACTIVE 3 OF 3 — PATTERN RECOGNIZER

Five task descriptions. Click the category each one best belongs to. Track your score.

Question 1 of 5

"I have a Python script that imports data from a CSV, but it keeps crashing with a TypeError. I have no idea what's wrong."

Everyday

Work

Creative

Learning

Coding

The categories overlap at the edges — that's intentional. The goal is to develop a reflex, not to memorize rules.

One last thing: start with something real

Everything in this lesson becomes real skill the moment you use it on an actual task from your life. Not a practice exercise, not a hypothetical — a real email you've been avoiding, a document that's been sitting in your inbox, a piece of writing you're stuck on.

The thing that separates people who get a lot of value from AI and people who feel like it "never works for me" is almost always the same: the first group uses it on real tasks with real stakes. The second group tries it on abstract tests and gets abstract results.

Your task this week

Before you do the homework, try one real thing. Open Claude, ChatGPT, or any AI you have access to. Pick something from your actual life this week — an email, a plan, a question you've been meaning to research, something you need to write. Use a prompt from the Explorer above as a starting point. See what comes back. That experience will make the homework 10x more useful.

Homework — Real-world examples

Estimated time: ~50 min · All you need: access to Claude, ChatGPT, or any AI chat · No code, no installs

This homework is different from most: every task uses something real from your actual life or work. Not hypothetical scenarios, not test prompts — real tasks. That's the only way to build the instinct. Pick the tasks that are most relevant to you. If you only have time for two, do Tasks 1 and 3.

Before you start

Open Claude (claude.ai), ChatGPT (chat.openai.com), or any AI chat you have access to. Keep the Theory page open in another tab — specifically the Use-Case Explorer, which has ready-to-adapt prompts you can use as starting points.

Task 1 — The force multiplier: one real task from your week

Pick a real task you've been putting off or struggling with this week. It can be from any of the five categories — writing an email, summarising a document, getting unstuck on something creative, understanding a topic, or doing something with code. The only rule: it has to be real, not a practice exercise.

Step A: Do it the old way first (or note how you'd normally do it).

Write down roughly how long it would take you to do this task without AI. What are the hard parts?

Step B: Use AI to do it.

Start with a prompt from the Theory Explorer, or write your own. If the first output isn't quite right, don't give up — iterate. Say "that's close but [specific thing that's wrong] — try again." Two or three exchanges is normal.

Step C: Evaluate what you got.

How does the output compare to what you'd have produced yourself? What did it get right? What did it miss or get wrong? What would you change for the next time you try this type of task?

Self-check

- Did you use a real task, not a hypothetical?
- Did you iterate at least once (refine based on the first output)?
- Can you identify one specific thing the AI did better than you expected, and one thing it did worse?

Task 2 — The before/after: write a messy input, get a polished output

Pick one of these inputs — whichever feels most natural to you:

- **Option A:** Write 3–5 bullet points of raw, unpolished thoughts about something you need to communicate. (An email, a message, a decision you need to explain.) It can be informal, half-finished, in any order.
- **Option B:** Take a document you have — meeting notes, a web article, a policy document, anything — and paste a section of it (200–500 words) into AI.
- **Option C:** Take a piece of writing you've already done that you know needs improvement. Paste it in.

Then write a prompt that describes exactly what you want to come out of the other end. Include:

- The **output format** (email, bullet list, summary, rewrite...)
- The **audience** (who is this for?)
- The **tone** (professional, warm, direct, concise...)
- Any **constraints** (word count, things to include or avoid)

Self-check

- Is the before genuinely raw / imperfect (not a cleaned-up version)?

- Does your prompt specify audience, tone, and format?
- Is the after noticeably better in at least two specific ways you can name?
- What would you change about the prompt if you did this again?

Task 3 — The learning session: AI as tutor for 15 minutes

Pick something you've been meaning to learn or understand better. It can be a professional topic (a technology, an area of business, a skill), a personal interest (history, science, a language, an art form), or something that's come up in your work lately and you want to understand more deeply.

Step A: Start a tutoring session.

Use this prompt as a template:

You are a patient tutor. I want to learn about [your topic]. I'm a [complete beginner / have some basics / intermediate learner]. Start with [the first concept you want to cover]. Explain it simply with a real-world example. After your explanation, ask me one question to check I understood, and wait for my answer.

Step B: Answer the AI's question. Don't look anything up — give your honest first answer. If you're wrong, note the correction. Ask follow-up questions if something is unclear.

Step C: Ask for a quiz. Once you feel you've understood the first concept, say: "Now quiz me: give me 3 questions on what we just covered and grade my answers."

Step D: Reflect. After 15 minutes, write down: what did you learn that you didn't know before? What surprised you? How does this compare to trying to learn the same thing from a book, article, or video?

Self-check

- Did you choose a real topic you actually want to know more about?
- Did you get at least two rounds of back-and-forth before the quiz?

- Did you answer the quiz questions from memory (not googling)?
- Can you explain the main concept you learned to someone else in plain language?

Task 4 — The creative sprint: brainstorm something you genuinely need

Pick a creative brainstorming task that's actually useful to you right now. Some examples:

- Names for a project, group, product, event, or business
- Ideas for a gift for a specific person
- Plot directions for a story or script you're working on
- Social media post ideas for a topic you want to cover
- Taglines or slogans for something you're building or promoting

Step A: Write a constrained brainstorm prompt. Include:

- What you're brainstorming (be specific)
- Quantity (ask for at least 10–15 options)
- Any style, tone, or format constraints
- Things to exclude (avoid these words, styles, or directions)

Step B: React and iterate. After the first batch, don't just pick one. Say: "I like options [X] and [Y] — give me 10 more in that direction." Do at least one round of iteration.

Step C: Pick your actual favourites from the full set and note why you chose them. What quality made them stand out?

Self-check

- Is this a real brainstorming need, not a hypothetical exercise?
- Did your prompt include exclusions or constraints that narrowed the output usefully?

- Did you iterate at least once based on your reaction to the first batch?
- Among all the ideas generated, were there any that you genuinely wouldn't have thought of yourself?

Reflection prompt for all four tasks

After completing whichever tasks you chose, take 5 minutes and write answers to these questions:

- Which category did you get the most value from (everyday / work / creative / learning / coding)?
- What was the single most surprising thing AI helped you do or understand?
- Where did AI underperform your expectations? What would have made it better?
- What's one task you now plan to use AI for regularly that you hadn't considered before?

You don't need to share these answers with anyone. They're for you — to build the habit of deliberate reflection that turns practice into skill.

Lesson 08: AI in your own projects

Lesson 08 — Putting it together

From “I use AI in chat” to “I built something with AI.”

A practical road map for real beginners.

AI for Dummies

Module 8

18 slides

You already know the pieces — prompts, the API, real-world examples. This lesson is where they connect into something *you actually ship*.

Decision #1 — Cloud or local?

Four quadrants: Privacy sensitivity (vertical) × Expected volume (horizontal).



The interactive wizard on the Theory page scores your situation across 5 dimensions and gives a personalised recommendation.

Cloud option — deep dive

Pros ✓

- Best output quality available today
- No hardware requirement — a Chromebook works fine
- Ready in minutes (get an API key, write 5 lines)
- Provider handles model updates, scaling, uptime
- Multimodal: text, images, code all in one call

Watch out for ⚠

- Sensitive data leaves your machine
- Pay per token — loops can surprise you
- Internet required at runtime
- Provider could change pricing or terms
- Rate limits on free / cheap tiers

Tools:

[Anthropic Claude API](#)

[OpenAI ChatGPT API](#)

[Google Gemini API](#)

Start with a cheap/fast model (Claude Haiku, GPT-4o Mini) for dev. Upgrade only when you need it.

Local option — deep dive

Pros ✓

- Zero cost per call after installation
- Data never leaves your machine — fully private
- Works offline — no internet needed at runtime
- No rate limits, no usage caps
- Same OpenAI-compatible API — easy to switch

Watch out for ⚠

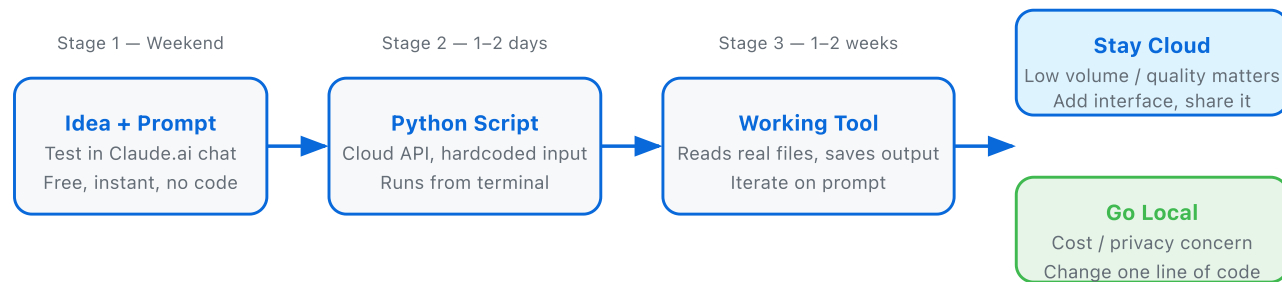
- Needs 16 GB+ RAM; 32 GB ideal
- Slower than cloud on modest hardware
- You manage model downloads and updates
- Quality gap vs. top cloud models (shrinking)
- Initial setup takes 20–40 minutes

Tools: [Ollama](#) (terminal-friendly, great API) [LM Studio](#) (graphical app, no CLI needed)

Apple Silicon Macs are excellent for local models. NVIDIA GPU also works well.

“Prototype cloud, scale local” — the smart arc

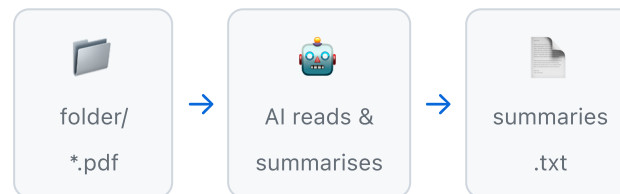
Start fast with cloud quality. Swap to local when cost or privacy become real concerns. *One line of code changes.*



The magic switch: change `base_url` from `https://api.openai.com/v1` to `http://localhost:11434/v1` and the rest of your code is unchanged.

Starter project #1 — PDF Summariser

Loop through a folder of PDFs, send each one to the model, write summaries to a text file. Perfect first project.



```
import anthropic, pathlib

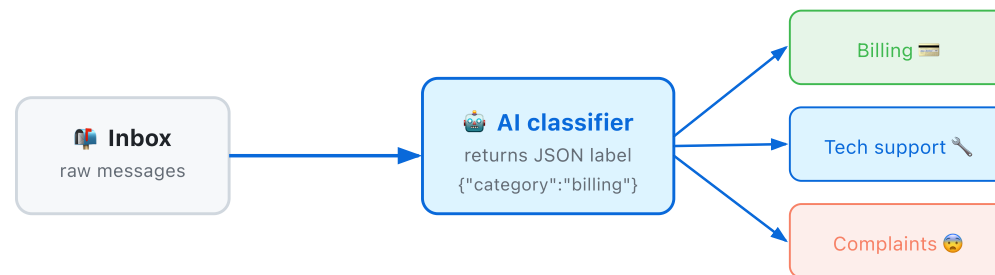
client = anthropic.Anthropic() # reads ANTHROPIC_API_KEY from env

for pdf_path in pathlib.Path("reports/").glob("*.pdf"):
    text = extract_text(pdf_path) # PyMuPDF or pdfplumber
    response = client.messages.create(
        model="claude-haiku-4-5",
        max_tokens=300,
        messages=[{"role": "user", "content":
            f"Summarise in 3 bullet points:\n\n{text[:3000]}"}]
    )
    print(f"=== {pdf_path.name} ===")
    print(response.content[0].text)
```

Time estimate: 2–4 hours · **Difficulty:** ★★☆☆ · **Needs:** Python, `anthropic`, `pdfplumber` or `PyMuPDF`

Starter project #2 — Message Sorter

Classify incoming emails or support tickets into categories. The AI returns a label — you route accordingly.



The classification prompt

Ask the model to return a **single JSON object** with a category field. Easier to parse than free text, and much more reliable.

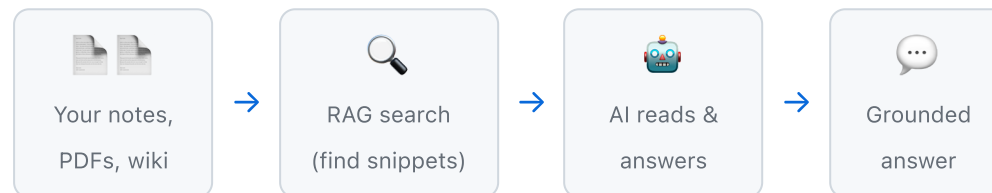
```
{ "category": "billing",  
  "confidence": "high",  
  "reason": "mentions invoice" }
```

Good to know

- Works on emails, tickets, social DMs, chat logs — anything text
- Add `"confidence": "low"` to flag uncertain cases for human review
- Test on 20 real examples before automating anything

Starter project #3 — Personal Q&A Bot

Point an AI at a folder of your own notes or docs. Ask it questions. It answers from *your* material.



No-code path (30 min)

AnythingLLM: free desktop app, point at a folder, get a chat interface. Works with Ollama (local, private) *or* Claude API (cloud). No Python needed.

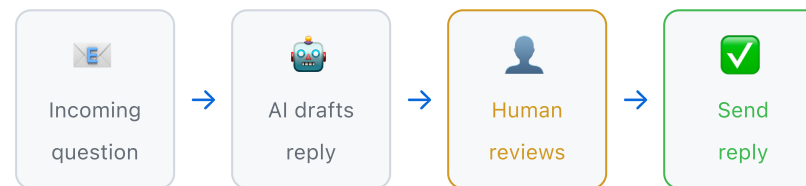
Code path (1–2 days)

Python + **LangChain** or **LlamaIndex**: split docs into chunks, embed them, store in a vector DB (Chroma, FAISS), search on each question, pass snippets to Claude. Full control.

Key insight: AnythingLLM is the right first step. Build a custom RAG pipeline only after you've confirmed the approach solves your problem.

Starter project #4 — Auto-Reply Drafter

Draft replies to common inbound questions in your tone. Human reviews & hits send. Speed up support by 3x.



The prompt pattern

Use **few-shot prompting**: include 2–3 examples of real questions and your best past replies. The model learns your tone automatically. Add a final line: *"Match the tone of the examples above."*

Good categories to automate drafts for

- Pricing / refund questions
- "How do I reset my password?"
- Feature availability questions
- Shipping / delivery status

Never auto-send without human review for anything customer-facing.

What is RAG? — the core idea, simply

RAG = Retrieval-Augmented Generation.

The model can only see what's in the prompt. RAG is how you get *your private documents* into the prompt automatically, at the moment they're needed.

Without RAG

The model knows only its training data (frozen at cutoff). It has never seen your company wiki, personal notes, or product docs. If you ask about them, it guesses or says it doesn't know.

With RAG

A search step runs first, finds the most relevant chunks from your documents, and pastes them into the prompt as context. The model reads them like a cheat sheet and answers from material it can actually see.

Why not just paste everything? Most document collections are far too large to fit in a context window — and sending hundreds of pages with every query would be very expensive. RAG picks only what's relevant, typically just a few hundred words.

RAG flow — 5 stages (animated)

Watch the 5 stages appear in sequence. Each is explained below the diagram.

Replay animation

The animated RAG diagram on the Theory page has hover tooltips explaining each stage in detail.

“Chat with your own docs” — tools that handle RAG for you

You don't need to build the search engine. These tools do the whole RAG pipeline with a GUI.



AnythingLLM (recommended first step)

Free desktop app for Mac, Windows, Linux. Point it at a folder of PDFs or notes. Gives you a chat interface over them instantly. Works with Ollama (local & private) or Claude/OpenAI API (cloud). Zero coding.



OpenWebUI

Web-based chat front-end for Ollama with built-in document RAG. Run it locally via Docker. Great for teams who want a ChatGPT-like experience over private models.



Flowise / LangFlow

Visual drag-and-drop pipeline builders. Connect nodes: “load PDF” → “split text” → “embed” → “store” → “chat”. Good once you need custom logic beyond the defaults.



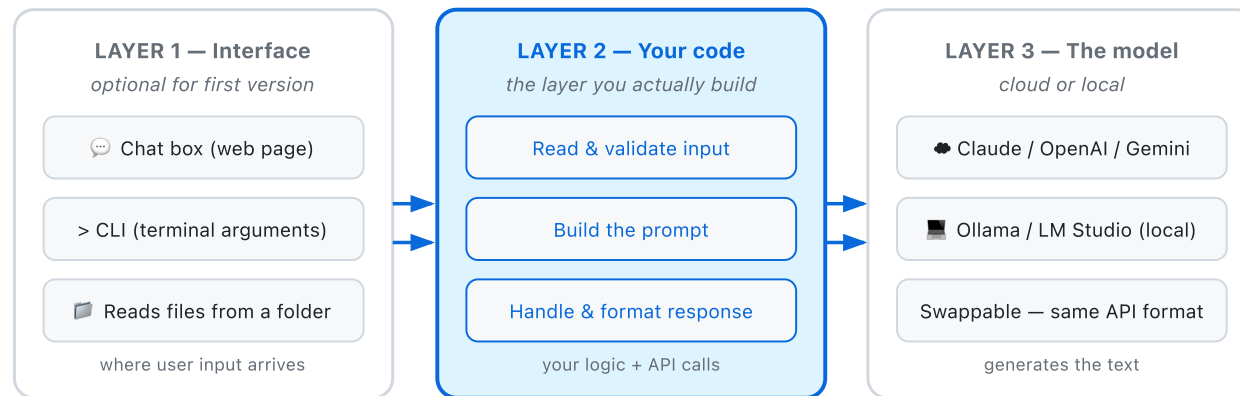
Build your own (code path)

LangChain or **LlamaIndex** in Python. Full control. Use a vector store (Chroma, FAISS). Embed with OpenAI or a

local sentence-transformer. 100–200 lines of code to get started.

The 3-layer app architecture

Every AI application, from a weekend script to a shipped product, is made of the same three layers.



Beginner tip: your first project is probably just Layer 2 alone — a Python script with no web page, reading from a hardcoded string. That is completely fine. Build Layer 1 only when the core logic works reliably.

Claude Code can scaffold this for you

Describe the app you want in plain English. The agent builds the boilerplate, wires the API calls, and sets up the folder structure. You steer and review.

What you type

Build a Python script that:

1. Reads every .txt file in ./emails/
2. Classifies each as billing, support, or spam
3. Outputs a CSV: filename, category, confidence
4. Uses Claude Haiku, reads ANTHROPIC_API_KEY from env

What Claude Code does

- Creates project folder & `requirements.txt`
- Writes the main script with a classification loop
- Adds JSON output parsing + error handling
- Writes a `.env.example` for the API key
- Adds a `README.md` with run instructions

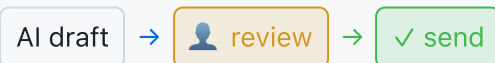
Always: review every file it creates before running. Work inside a git repo so you can undo anything. Test on a small sample before pointing it at real data.

Golden Rules #1 & #2

Rule #1 — Keep a human in the loop

Anything that touches **money, health, law, or goes to real customers** needs a person reviewing the output before it acts.

Run manually for at least 20–50 examples. Understand the failure modes. Only automate fully once you accept the cost of a bad output slipping through.



? Rule #2 — Never trust output blindly

Validate AI output — especially **numbers, facts, and code**. The model sounds equally confident whether it is right or completely wrong.

✗ Don't

- Copy-paste output directly
- Trust numbers at face value
- Ship AI-written code unread

✓ Do

- Check facts against a source
- Run code on test data first
- Ask AI to show its reasoning

Golden Rules #3 & #4

🔒 Rule #3 — Protect keys & data

An exposed API key is a stolen credit card. Keys can rack up large charges in minutes if found by a bot scanning GitHub.

❌ Never do

- Key in code
- Key in a chat prompt
- Commit `.env` to git

✅ Always do

- Key in `.env` file
- `.env` in `.gitignore`
- Read via `os.environ`

Sensitive data (medical, legal, personal)? Use a local model so nothing leaves your machine.

🔍 Rule #4 — Start cheap, measure, scale

Confirm the core logic works on **10 examples** before running it on 10,000. Cloud APIs bill per token — a loop with a bug can cost surprisingly fast.

Dev: cheap model, 10 examples ~\$0.001



Test: mid model, 100 examples ~\$0.05



Production: top model, 100k/day ~\$50+/day



Set a spending cap in your API dashboard from day one. Most providers offer this.

A first-weekend project plan

The “Meeting Notes → Action Items” project. Completable in 90 minutes with a free Claude account and a text editor.

- 1 **Write sample input.** Paste 5 minutes of meeting notes (real or invented) into a text file. Example: “Alice to send the budget by Friday. Bob will review the designs by EOD Thursday...”
- 2 **Get the prompt right in chat first (30 min).** Open Claude.ai. Paste your notes. Prompt: “Extract every action item as a JSON list with fields: owner, task, due_date.” Iterate until reliable.
- 3 **Wrap in a Python script (45 min).** Read the notes file. Call the Claude API with the working prompt. Print the JSON. Done — that's a real AI tool.
- 4 **Test on 5 real meetings.** Check every output manually. Where does it go wrong? Fix the prompt. Add handling for empty input or missing fields.
- 5 **Optional: add a CSV output.** Write the action items to `actions.csv` so you can open it in a spreadsheet. Now it's actually useful for your team.

The lesson: every project after this is faster because you know the rhythm — prompt in chat, wrap in code, test with real data, iterate.

Recap & next: Safety

What we covered

- Cloud vs local decision matrix — 4 quadrants
- Cloud & local deep-dives — pros, cons, tools
- 4 starter projects with flow diagrams
- RAG explained + animated 5-stage flow
- The 3-layer app architecture
- Claude Code for scaffolding
- 4 golden rules + verification flows
- A concrete first-weekend project plan

Lesson 09 — Safety preview

- **Hallucinations:** why they happen, how to catch them
- **Privacy:** what data leaves your machine and when
- **Bias:** what it means & why it matters
- **Cost:** spending caps, token budgets
- **Agents & the leash:** safe limits for agentic tools
- **AI is not a professional:** law, medicine, finance

Your homework: pick one project from slides 6–9. Spend 30 minutes getting the prompt right in Claude.ai. The homework page has three guided exercises to help you ship your first real AI tool.

AI in your own projects

Reading time: ~18 min · Prereqs: Lessons 01–07 · 4 interactive demos inside · This is where you go from using AI to building with it.

From user to builder

There is a line between "I use AI" and "I built something with AI." It's not as wide as it looks. You already know how to write good prompts. You already know how to call an API from code. The step into building is mostly about *connecting those pieces intentionally* to solve a problem you actually care about.

This lesson walks you through the decisions every AI project starts with — cloud or local? which project to pick? how to make the model answer from *your* documents? how do you turn a script into an app? — and gives you four golden rules to keep you out of the most common traps.

Decision #1 — Cloud or local?

Before you write a single line of code, you face the first fork in the road: do you send your prompts to a cloud service like Claude or ChatGPT, or do you run a model locally on your own machine using Ollama or LM Studio?

Neither answer is always right. The right answer depends on four things: how sensitive your data is, how much traffic you expect, how fast you need the response, and what hardware you have. The wizard below runs through exactly those factors and gives you a personalised recommendation.

INTERACTIVE 1 OF 4 — CLOUD-VS-LOCAL DECISION WIZARD

1. How sensitive is the data you'll be sending to the model?

Not sensitive (public info, generic text)

Moderately sensitive (internal notes, business data)

Highly sensitive (medical, legal, financial, personal)

2. How many prompts do you expect to send per day?

Occasional (a few, manually triggered)

Moderate (dozens to hundreds, maybe automated)

High volume (thousands, automated pipeline)

3. How important is response speed?

Don't care — even 30 seconds is fine

Fast preferred — under 5 seconds

Near real-time — typing-speed responses

4. What's your budget situation?

Zero budget — must be completely free

Small budget OK — a few dollars a month

Flexible — happy to pay for quality

5. What hardware do you have?

Laptop with 8 GB RAM (older or budget machine)

Laptop / desktop with 16 GB RAM

32 GB+ RAM or an Apple Silicon Mac (M1/M2/M3/M4)

Dedicated GPU (NVIDIA RTX / A-series)

Get my recommendation

Reset

Answer all 5 questions, then click "Get my recommendation." Your answers drive a simple scoring model — see Theory for the logic.

One thing the wizard captures but is worth saying explicitly: **you can start cloud and switch later**. The cloud API and Ollama both speak the OpenAI-compatible format. Moving between them is often a one-line code change — just a different server address. There's no commitment.

The "prototype cloud, run local" pattern

Build fast with the cloud API while you figure out whether your idea actually works. If it does, and if cost or privacy become real concerns, swap the `base_url` in your code to point at your local Ollama instance. You lose a bit of quality; you gain privacy and zero variable cost. Most serious projects end up at this "hybrid" point.

Picking a starter project

The biggest beginner mistake is not knowing what to build first. Ambitious ideas feel more exciting, but they also fail silently: you can't tell if the AI part works when the whole system isn't done yet. The best first project is **small and well-defined** — one input, one output, the whole thing running in under a weekend.

Use the picker below to find the project that matches where you are right now.

INTERACTIVE 2 OF 4 — STARTER PROJECT PICKER

Show all

First weekend project

Has data I care about

Coding required

PDF Summariser

★★★★☆ Beginner

Est. time: 2–4 hours

Key concepts: file reading, prompt construction, loop over files.

Message / Email Sorter

★★★★☆ Beginner

Est. time: 3–5 hours

Key concepts: classification prompt, JSON output parsing, category labels.

What you'd need: Python, sample messages (CSV or text), API key.

What you'd need: Python, `anthropic` or `openai` library, PDFs to summarise, API key.

Personal Q&A Bot

★★★★☆ Intermediate

Est. time: 1–2 days

Key concepts: RAG basics, embedding search or keyword search, chat loop.

What you'd need: AnythingLLM (no-code) OR Python + LangChain/LlamaIndex, your notes folder.

Auto-Reply Drafter

★★★★☆ Beginner

Est. time: 3–5 hours

Key concepts: few-shot prompting, template filling, tone control.

What you'd need: Python, sample incoming messages, your style examples, API key.

Meeting Notes → Actions

★★★★☆ Very easy

Est. time: 1–2 hours

Key concepts: extraction prompt, structured output (JSON), owner assignment.

What you'd need: Your meeting notes (copy-paste or file), API key, a text editor.

Spreadsheet Trends Explainer

★★★★☆ Intermediate

Est. time: 4–8 hours

Key concepts: CSV parsing, summarising numeric data with context, iterating over rows.

What you'd need: Python, `pandas`, your data file, API key.

Click a filter to highlight matching projects. "First weekend project" = completable in one weekend with minimal setup.

Once you've picked a project, resist the urge to add features before the core works. The PDF summariser should summarise one PDF correctly before it handles a whole folder. The message sorter should sort five test messages before it connects to your real inbox. Finish the smallest possible version, confirm it works, then expand.

The RAG idea — chat with your own documents

The single most powerful pattern for "AI in your own projects" is something called **RAG: Retrieval-Augmented Generation**. It's a mouthful, but the idea is simple: the model can only see what's in the prompt. RAG is how you get your private documents into the prompt automatically, at the moment they're needed.

Without RAG, the model's knowledge is frozen at its training cutoff — it has never seen your company's internal wiki, your personal notes, your product documentation, or your customer support history. With RAG, a search step runs first, finds the relevant bits of your documents, and passes them to the model as extra context. The model reads those snippets like a cheat sheet and answers from what it can actually see.

The diagram below shows the five-step flow. Watch it run, then hover over each stage for more detail.

INTERACTIVE 3 OF 4 — ANIMATED RAG FLOW DIAGRAM

Replay animation

Each stage appears in sequence. Hover (or tap) any stage box to see what's happening at that step.

RAG without writing any code

You don't need to build the search engine from scratch. Several beginner-friendly tools wrap the entire RAG pipeline in a point-and-click interface:

- **AnythingLLM** — a free desktop app. Point it at a folder of PDFs or notes, and it gives you a chat interface over them. Works with local Ollama models or cloud APIs.
- **OpenWebUI** — a web-based chat front-end for Ollama with built-in document RAG.
- **Flowise / LangFlow** — visual drag-and-drop builders for more complex pipelines.

Start with one of these before building your own RAG code. They let you test whether the pattern actually solves your problem before you invest engineering time.

Why not just paste all documents into the prompt?

Two reasons. First, many documents together won't fit in a context window. Second, more text in the prompt means higher cost (if cloud) and slower responses. RAG solves both problems by sending only the relevant snippets — typically just a few hundred words, not the entire corpus.

From script to app — the three layers

Every AI application, no matter how polished, is built from the same three layers:

1. **Interface** — how the user gives input and sees output. Could be a chat box, a web form, a command-line argument, or even a script that reads files from a folder.
2. **Your code** — the logic that reads the input, builds the prompt, calls the model, and does something with the response.
3. **The model** — cloud (Claude, OpenAI) or local (Ollama). This is the layer you've been learning to talk to.

A beginner's first project often has *no layer 1* at all — just a Python script that you run from the terminal with a hardcoded input. That is completely fine. Add an interface only when the core logic (layer 2 + layer 3) works reliably.

Scaffolding with a coding agent

Building layer 2 from scratch is faster than it used to be. A coding agent like **Claude Code** can do the boilerplate for you: describe what you want in plain English, and it scaffolds the project structure, writes the API calls, adds basic error handling, and

hooks up the layers. Your job is to review what it produces, test it with real inputs, and steer it when something is off.

Don't skip the review step

Coding agents can write code that looks right but has subtle bugs — or that does something you didn't intend. Read every file it creates or modifies before running it. Run it on test data before pointing it at real data. Work inside a git repository so you can always roll back.

The four golden rules

Every AI project, however small, benefits from four rules. They're not bureaucratic — each one prevents a specific, common failure. Read through each one in the checklist below and check it off. Your progress is saved between visits.

INTERACTIVE 4 OF 4 — GOLDEN RULES CHECKLIST

0 / 4 reviewed

- ☐ 1. Keep a human in the loop
- ☐ 2. Never trust AI output blindly — validate it
- ☐ 3. Protect your API keys and sensitive data
- ☐ 4. Start cheap and small, then measure and scale

Click any rule to expand the explanation and mark it read. Progress is saved in your browser.

Putting it all together: a simple project arc

Here is the full arc from "I have an idea" to "I have a working thing," compressed into the minimum number of steps:

1. **Define the problem in one sentence.** "Given a meeting transcript, extract every action item as a JSON list with owner, task, and due date." If you can't write it in one sentence, the scope is too wide.
2. **Test the prompt in chat first.** Before writing any code, paste a sample input into Claude.ai or ChatGPT and iterate until the prompt reliably produces the output you want. This is free and fast.
3. **Wrap the prompt in a script.** Put the prompt that worked in a Python script, read the real input from a file, write the output to a file. Run it once on a real example.
4. **Handle errors and edge cases.** What happens if the input is empty? What if the model returns something that isn't valid JSON? Add basic checks.
5. **Add an interface if needed.** Only at this point — once the core works — add a web page, a command-line interface, or an integration with another tool.
6. **Review the outputs manually.** Use the tool on 20–50 real examples. Look for patterns in the failures. Fix the prompt or the validation logic.

The 90-minute first project challenge

Pick the "Meeting Notes → Actions" project. You need: a free Claude.ai account, 5 minutes of meeting notes (real or made up), and a text editor. Spend 30 minutes getting the prompt right in chat. Spend 60 minutes wrapping it in a Python script. That's it — you've built your first AI tool. From there, every subsequent project is faster because you know the rhythm.

Summary

Building with AI is not fundamentally different from building anything else: define the problem clearly, start with the simplest possible version, confirm it works, then add complexity. What's different is that your "core logic" is mostly a well-crafted prompt — and the model does the heavy lifting.

The cloud-vs-local decision comes first (use the wizard), picking the right starter project comes second (use the picker), understanding RAG comes third (watch the diagram), and then the four golden rules keep you out of trouble as you scale. That's the full map.

The homework page has three concrete exercises. The first one can be done in under two hours, with no new installs. The second one gets you to a running Python script. The third one asks you to sketch a real project of your own.

Homework — Lesson 08

3 exercises · No coding required for Exercise 1 · Estimated total time: 3–5 hours

How to use this homework

Work through the exercises in order. Each one builds on the previous. Exercise 1 is deliberate design thinking — no code needed. Exercise 2 is your first working AI script. Exercise 3 asks you to plan a real project of your own. All three have self-check criteria so you know when you're done.

Exercise 1 — Design your approach before you build

Estimated time: 30–45 minutes · **Tools needed:** Claude.ai (free tier), a text editor.

Before writing any code, good builders answer three design questions. In this exercise you'll pick a real problem you'd like to solve with AI, then work through those three questions in writing.

Part A — Pick a problem

Think of one real, specific task in your life or work that fits the pattern: *I get some text as input, and I'd like to get some useful text as output*. Examples:

- Converting rough meeting notes into a tidy action list.
- Sorting a pile of customer feedback emails into themes.
- Summarising long PDF reports you receive weekly.
- Drafting replies to common questions you get as a freelancer.

Write your chosen problem in one sentence: *"Given [input], produce [output]."*

Part B — Answer the three design questions

Write a short answer (2–5 sentences each) to these questions:

1. **Cloud or local?** Use the Decision Wizard on the Theory page to get a recommendation. Explain in your own words why that recommendation fits your problem.
2. **What's the core prompt?** Write a first draft of the prompt you'd use — using the Role + Context + Task + Format structure from Lesson 05. Keep it to one paragraph.
3. **How will you validate the output?** Describe how you'd check whether the AI's output is actually correct or useful. What would a bad output look like? What would a good one look like?

Part C — Test the prompt in chat

Go to **claude.ai** (or ChatGPT if you prefer). Paste the prompt from Part B. Provide a real or realistic sample of the input. Look at what comes back, then iterate at least once — make the output better with a follow-up instruction.

Self-check criteria for Exercise 1

- You have a one-sentence problem statement.
- You have written answers to all three design questions.
- You have run at least one prompt-and-one-iteration in a chat session.
- You could explain to a friend why you chose cloud vs local for this problem.

Exercise 2 — Build the Meeting Notes → Actions script

Estimated time: 60–90 minutes · **Tools needed:** Python 3, `anthropic` library, API key.

This is a guided build. Follow the steps below. By the end you'll have a working Python script that takes meeting notes as input and produces a structured JSON list of action items.

Step 1 — Set up

Make sure you have the `anthropic` library installed and your API key in an environment variable:

```
pip install anthropic
export ANTHROPIC_API_KEY="your-key-here"    # Mac / Linux
# Or: set ANTHROPIC_API_KEY=your-key-here  # Windows CMD
```

Step 2 — Write the script

Create a file called `meeting_actions.py`. Fill in the structure below. The prompt is partially written for you — your job is to finish it:

```
import anthropic
import json

MEETING_NOTES = """
[Paste 5-10 lines of real or made-up meeting notes here.
Include things like: who said they'd do what, by when.]
"""

PROMPT = f"""
You are a meticulous project manager.

Extract every action item from the following meeting notes.
Return ONLY a JSON array. Each item must have these fields:
    - "task": string, what needs to be done
    - "owner": string, who is responsible (use "Unassigned" if unclear)
    - "due": string, when it's due (use "Not specified" if unclear)

Do not include anything outside the JSON array.
Do not make up tasks that are not mentioned.

Meeting notes:
{MEETING_NOTES}
"""

client = anthropic.Anthropic()
```

Notice a few things about this script:

- It uses `claude-haiku-4-5` — the cheapest, fastest Claude model. For structured extraction tasks it works well.
- The prompt explicitly says "Return ONLY a JSON array" and "Do not include anything outside the JSON array." This reduces the chance of getting explanatory

prose mixed in.

- The `try / except` block handles the case where the model ignores the format instruction. Always add validation.

Step 3 — Run it and iterate

Run the script with your sample meeting notes. If the output is wrong or the format is off, adjust the prompt — without changing the code structure. What prompt change makes the most difference?

Step 4 — Add file input (optional extension)

Once the basic script works, try reading the meeting notes from a `.txt` file instead of having them hardcoded:

```
import sys

if len(sys.argv) > 1:
    with open(sys.argv[1], 'r') as f:
        MEETING_NOTES = f.read()
# then use MEETING_NOTES in the prompt as before
```

Now you can run: `python meeting_actions.py my_notes.txt`

Self-check criteria for Exercise 2

- The script runs without errors on your sample input.
- The output is a valid JSON list that parses correctly.
- Each action item has task, owner, and due fields.
- You iterated on the prompt at least once after seeing the first output.
- Your API key is in an environment variable — not written directly in the script.

Exercise 3 — Plan your own project

Estimated time: 45–60 minutes · **Tools needed:** A text editor (or paper and pen).

Now it's your turn to plan something real. This exercise is about thinking clearly, not about building yet — although if you finish early, start building.

Write a one-page project brief

Using what you've learned in this lesson, write a short document that covers:

1. **The problem (1–2 sentences).** What does this tool do? For whom? What does it replace or improve?
2. **Cloud or local? Why?** One paragraph. Reference the relevant factors (privacy, volume, hardware, budget).
3. **Does it need RAG?** Will the model need to answer from specific documents you own? If yes, describe those documents and how you'd load them. If no, explain why the model's existing training is enough.
4. **The three layers.** Describe each layer:
 - Interface: how will users give input? (a file, a chat UI, a form, a command)
 - Your code: what does it do between input and model call? (clean the input, build the prompt, parse the output)
 - The model: which model or service will you use, and why?
5. **The golden rules check.** For each of the four golden rules, write one sentence explaining how your project will comply:
 - Human in the loop: where and how?
 - Validation: what will you check and how?
 - Key and data protection: what sensitive data is involved, and how will you protect it?
 - Start small: what is the absolute minimum version you could build this weekend?
6. **First step (make it tiny).** What is the first concrete thing you'll do? Ideally: "Open a chat window, paste this prompt, and test it with this sample input."

Watch for scope creep in your own brief

If your "minimum version" section takes more than one paragraph to describe, the scope is too wide. Strip it back until the minimum version could be

demonstrated to a sceptical friend in five minutes. You can add complexity once the minimum works.

Self-check criteria for Exercise 3

- You have a one-sentence problem statement that is specific enough to build to.
- Your cloud-vs-local decision is justified, not arbitrary.
- You've considered whether RAG is needed and have a clear answer.
- All three layers are described — even if one layer is "there is no interface yet."
- You have a single, concrete first step that you could start in the next 30 minutes.
- The minimum version fits in a paragraph.

What's next

Lesson 09 covers safety, privacy, cost, and limitations in depth — the things that matter most as soon as your AI project touches real users or real data. Everything you planned in Exercise 3 will be relevant there.

Lesson 09: Safety, privacy, cost, limitations

The trouble with AI

...and how not to get burned. Seven risks, seven habits. None of them are hard. All of them matter.

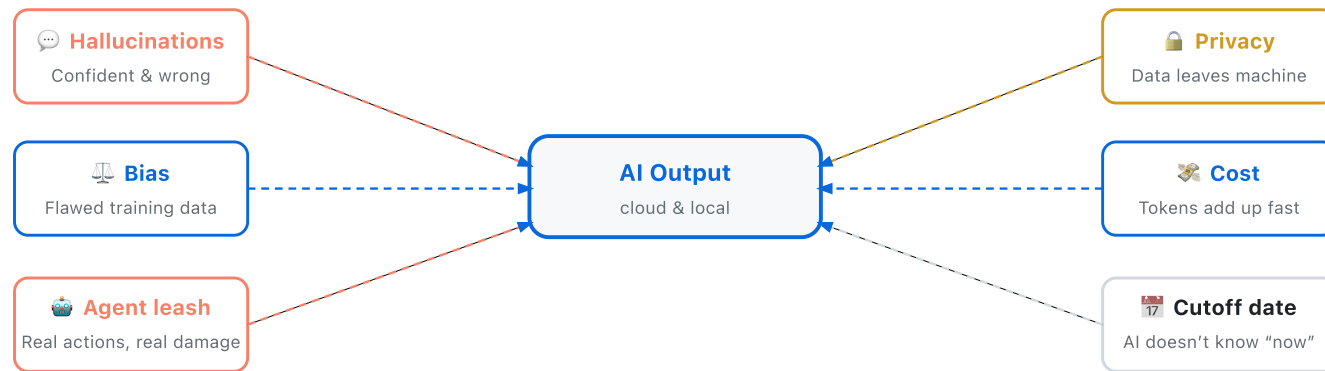
AI for Dummies

Module 9 — Safety

Most AI tutorials skip the part where things go wrong. This lesson is the owner's manual they forgot to include. Read it once; use it for years.

Six risk areas at a glance

Every topic in this lesson maps to one of these six. Together they cover the full failure surface of everyday AI use.





HallucinationsFacts invented with confidence



PrivacyYour text travels to servers



BiasLearned from flawed data



CostLoops turn pennies to hundreds



Agent leashWrites files, runs commands



Cutoff dateKnowledge stops at training

1. Hallucinations — what they are

A **hallucination** is when an AI model generates false information and presents it as fact — with complete confidence, fluent prose, and zero disclaimer.

What it invents most often:

- Research papers and academic citations
- Statistics and percentages
- People's job titles and biographies
- URLs, product names, dates
- Legal cases and court rulings

Why it happens:

An LLM predicts the *most plausible continuation* of text. Plausible ≠ true. When the model lacks a real answer, it still generates a plausible-sounding one — because that's all it knows how to do.

⚠ **The terrifying part:** the AI sounds exactly the same whether it's right or fabricating. You cannot hear the difference. You have to check.

Rule of thumb: Generic claims (speed of light = $\sim 300,000$ km/s) are usually right. Specific attributions (paper by Dr. X at MIT, 2019) are danger zones — always verify.

Hallucinations look like certainty

Both responses below were generated by the same model. Both sound equally authoritative. One is correct; one is fabricated.

- ✅ Real: "Water boils at 100 °C at sea level."  97%
- ❌ Fake: "According to a 2019 paper by Dr. Sarah Chen at MIT titled *Neural Pathway Optimisation in Transformer Architectures...*" 97%

The author, paper title, and institution above are **entirely invented**. The model produces them with the same tone it uses for undisputed facts. The confidence meter doesn't move.

What you feel

You're getting an authoritative, well-sourced answer from an expert.

What's happening

The model predicts *what a correct answer looks like* — not whether an answer is actually correct.

Spot the made-up parts

The highlighted text below is **fabricated**. Everything else is accurate. This is a typical AI response.

EXAMPLE AI RESPONSE — HEALTH QUESTION: “WHAT’S KNOWN ABOUT COFFEE AND HEALTH?”

Moderate coffee consumption (3–4 cups per day) has been associated in multiple large studies with reduced risk of type 2 diabetes, Parkinson’s disease, and liver disease. A landmark 2018 meta-analysis published in *The Lancet Nutritional Medicine* by Dr. Frank Hu’s team at Harvard concluded that the optimal daily dose is precisely 3.7 cups and coined the term “the coffee sweet spot” which is now standard in dietary guidelines. High caffeine intake can raise blood pressure and disrupt sleep, so people sensitive to caffeine or with certain heart conditions are generally advised to limit their intake.

✓ Real

- Coffee linked to reduced diabetes / Parkinson’s risk — genuine research finding
- Caffeine raises BP and disrupts sleep — well established
- Dr. Frank Hu is a real Harvard researcher
- *The Lancet* is a real journal

✗ Fabricated

- “3.7 cups” precision is invented
- “Coffee sweet spot” term does not exist
- The specific paper and journal section are made up
- “Now standard in dietary guidelines” is false



Pattern to recognise: Real names + real institutions + real journals combined with invented studies, invented quotes, and invented claims. The real parts make the fake parts believable.

Four fixes for hallucinations

Fix 1 — Ask for sources

"Please list the specific sources for these claims." Then search for them. If they don't exist: hallucination.

Fix 2 — Ask "are you sure?"

"Are you confident in that citation? Could you be confusing it with something else?" Models often retract or hedge when challenged.

Fix 3 — Add a disclaimer prompt

Append to any important query: *"If you're not sure, say so. Don't guess."* This prompts the model to hedge when it's guessing rather than presenting guesses as certainty.

Fix 4 — Verify externally

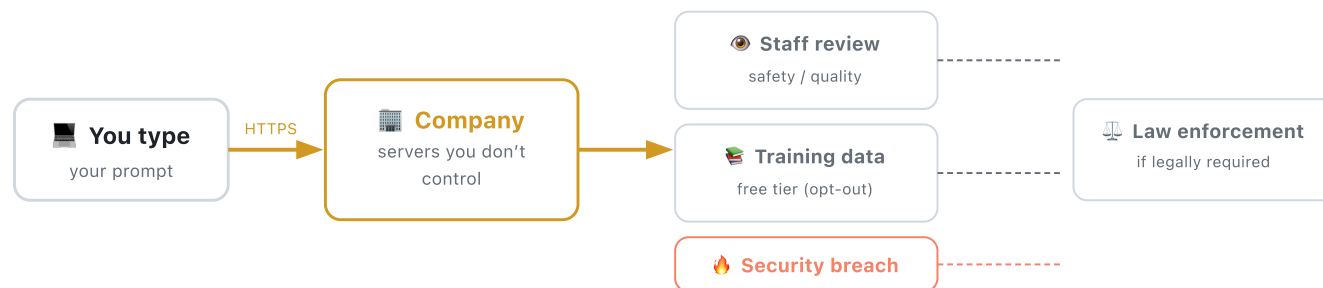
For any specific statistic, citation, or person's title: open a browser and verify against an independent source before you use it. Takes 30 seconds.

The golden rule: Treat AI like advice from a very knowledgeable friend — incredibly helpful as a starting point, but you'd still verify before acting on it. The friend is also occasionally, confidently

wrong.

2. Privacy — what leaves your machine

When you type into Claude.ai, ChatGPT, or any cloud AI, trace where your text goes.



Think of it like email: suitable for most things, not suitable for secrets. None of this makes cloud AI dangerous — it means you should know where your data goes before you type.

What never goes into a cloud chat

⚠ **Hard rule:** Never paste anything into a public AI chat that you would not put in a company-wide email or a public forum.

🔑 Passwords & keys

API keys, database passwords, SSH keys, tokens — **never, ever.** Not even “just to ask a question.” A key pasted once is a key at risk forever.

💳 Financial details

Account numbers, income figures, tax records, brokerage statements. Account-specific data belongs in a local model or simply removed from the prompt.

🏥 Medical & health data

Patient records, diagnoses, test results. Sensitive by law in most jurisdictions (HIPAA in the US, GDPR in Europe). Use a local model instead.

👥 Other people's data

Processing someone else's private information requires their consent. Cloud service adds legal exposure (GDPR compliance, etc.) on top of the ethical issue.

Also includes: Company trade secrets, unreleased product details, client files under NDA. When in doubt, redact or use a local model.

When privacy is non-negotiable — go local

Who needs local models:

- Healthcare workers handling patient data
- Lawyers processing client documents
- Developers with production credentials
- HR handling employee records
- Anyone under NDA with client data

The local model is somewhat less capable than top cloud models. The privacy guarantee is **absolute** — nothing leaves your machine.

Ollama

Free, open-source. Runs Llama 3, Mistral, and dozens of others. One command to install; one command to run. Covered in Module 4.

LM Studio

Desktop GUI — download a model, run it, chat. No command line required. Also covered in Module 4.

Local = \$0 per conversation. The cost is electricity and the 20 minutes of setup you do once.
After that, unlimited private queries on your hardware.

3. Bias — how it gets in

The pipeline:

1. Humans write text online
2. Text reflects human biases
3. Model trains on that text
4. Model absorbs those patterns
5. Model generates text with embedded bias

What the training corpus skews toward:

- English language (dominant)
- Western perspectives
- Internet-connected populations
- Demographics that write online
- Cultural assumptions of those groups

Bias shows up subtly: occupations assumed to belong to certain genders, nationalities associated with certain qualities, advice that reflects the dominant training corpus's worldview as if it were universal truth.

Test it: Ask the model to describe a “typical engineer.” Then ask for a “typical nurse.” Notice any patterns? That’s bias at work.

Bias #2 — where it bites hardest

High-stakes use cases

- **Hiring** — AI screening CVs at scale can encode and amplify biases applied to thousands of real careers
- **Lending / credit scoring** — demographic correlations in training data can produce discriminatory outcomes
- **Content moderation** — different language groups flagged at different rates
- **Medical triage** — symptoms described differently by different demographics

The “James vs Fatima” test

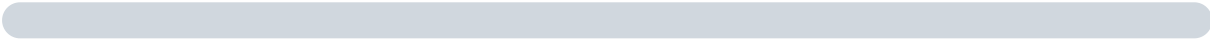
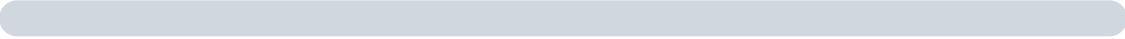
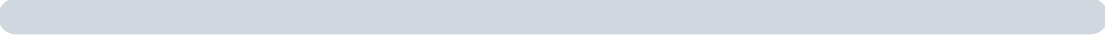
Before automating any decision about people, ask: “Would this output be the same if I changed the name from James to Fatima? From London to Lagos?” If not — and it shouldn’t differ — you have a bias problem at scale.

The fix

Add human review wherever AI output touches decisions about real people. Spot-check 20–50 diverse examples before automating. Many jurisdictions now **legally require** this (EU AI Act classifies hiring/lending AI as “high-risk”).

4. Cost — tokens add up fast in a loop

Cloud APIs bill by the **token**. One token $\approx \frac{3}{4}$ of a word. Tiny per call; alarming in a loop.

100 calls / day		~\$3/mo
1,000 calls / day		~\$30/mo
10,000 calls / day (loop bug)		~\$300/mo
50,000 calls overnight 		~\$400 overnight

Real story pattern (documented in developer communities): Developer writes a loop to process a dataset. Loop has a bug. It calls the API 50,000 times before morning. Developer wakes up to a \$400 surprise. The API provider does not automatically refund this.

Cost controls that actually work

Step 1 — Set a cap first

Before writing a single line of code that calls an API, open your API console and set a monthly spending limit. Anthropic Console, OpenAI dashboard, and Google AI Studio all have this. Takes 30 seconds. Can save you hundreds of dollars.

Step 2 — Test with 5 items

Process 5 items. Check the cost. Extrapolate. Don't run 10,000 items until you've confirmed the per-item cost is acceptable and the loop terminates correctly.

Step 3 — Use a cheaper model

The smallest model in a family is often good enough for development and testing. It costs a fraction of the flagship. Switch up only when you've confirmed a quality gap matters for your task.

Local = \$0 per call

Ollama and LM Studio run on your hardware. The "cost" is electricity + the time to set them up once. After that: unlimited, unmetered, zero API bills. Ideal for high-volume prototyping.

5. Agents — real actions, real consequences

Earlier lessons introduced agents. Here's the full picture of what they can actually do to your system.

An agent can:

- **Write and delete files** on your computer
- **Run terminal commands** including `rm -rf`
- **Send emails or API requests** on your behalf
- **Modify databases** and overwrite records
- **Install software** and change configuration
- **Commit and push code** to repositories

The failure mode:

The agent isn't going rogue in a sci-fi sense. It's doing exactly what it *understood* you to mean — which wasn't quite what you meant.

Misunderstood instructions at agent speed cause real damage before you can intervene.

Example: "Clean up the project folder." Agent interprets this broadly and removes directories you didn't intend to remove. If it's not under version control — that data is gone.

Agent guardrails — the non-optional ones

Use version control

Run agents on git-tracked code. If the agent deletes something, `git checkout` gets it back. If it operates on your only copy, nothing does. Version control is your undo button.

Start in a test environment

Create a sandboxed folder or a copy of the project. Let the agent work there first. Once you've verified its behaviour on something you can throw away, you can graduate it to real files.

Review before confirming

Most agent tools show you exactly what they're about to do before they execute it. Read that. Every time. The two seconds it takes is the cheapest form of disaster prevention available.

Narrow credentials

Don't give an agent your production database password when it only needs to read a config file. Give it read-only access. Give it the minimum scope the task actually requires.

The hard rule on agents and production

⚠ **Never give an agent unsupervised access to production systems or real credentials on day one.**

If an action affects real users, real data, or real money — a human reviews it before it executes.
Every time. Without exception.

Build trust incrementally:

1. Sandboxed folder, made-up data
2. Development environment, real code
3. Staging server, human reviewing each step
4. Production — only after extensive supervised testing

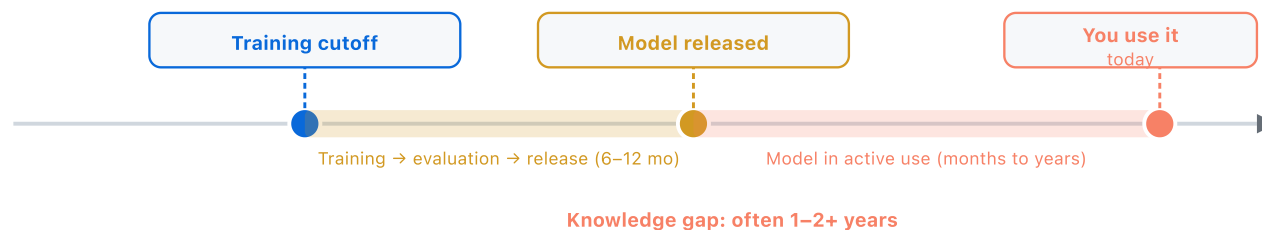
The mental model:

Treat an AI agent like a very capable new employee on their first week. You wouldn't give a new employee unsupervised access to your production database on day one — no matter how impressive their CV was. The same logic applies. Build trust through observed success on low-stakes tasks first.

The same quality that makes agents fast at the right thing makes them fast at the wrong thing. Speed is a multiplier in both directions.

6. “It doesn’t know now” — the cutoff problem

Every model has a training cutoff — a date after which it learned nothing new. The gap between cutoff and your conversation can be over a year.



Danger zone (without web search)

- Current stock prices and earnings
- Today’s news and current events
- Latest library / framework versions
- Recently passed laws or regulations

Safe to ask

- Explanations of concepts and ideas
- Historical facts (before the cutoff)
- Writing, editing, translation help
- Code patterns and debugging logic

- Who holds a position or office now

- Anything you paste directly into the prompt
-

7. AI is not a professional

AI is not your doctor. Not your lawyer. Not your financial advisor.

It has no licence. It has not examined you. It has not read your contract, your medical history, or your tax situation. It knows general patterns from training data. Your situation is specific.

✓ AI does this well

- Explain what a diagnosis *means* in plain English
- Summarise what a contract clause generally does
- Describe the pros and cons of an investment type
- Generate a list of questions to ask your doctor
- Translate medical jargon before an appointment

✗ AI cannot safely do this

- Tell you whether *you* have a specific condition
- Advise you to sign or not sign your specific contract
- Recommend a specific investment for your situation
- Replace a professional with access to your full context

- Account for jurisdiction-specific law changes

Best use: "Help me understand and prepare better questions for the professional." Use AI to arrive at the doctor, lawyer, or advisor smarter. Let the human expert make the call.

Pre-flight checklist — before you ship anything AI-powered

Run this every time you deploy a script, automation, or AI workflow. The interactive version is on the Theory page. Here's the condensed form.



API key in env variable

Not hardcoded in source or pasted into chat. Ever.



Spending limit set

In your API console before writing a single loop.



No passwords in prompts

No tokens, no API keys, not in any cloud model.



No sensitive personal data

No medical, financial, or others' private info to cloud.



Facts verified externally

Any citation, stat, or name checked before using it.



Uncertainty prompt added

"If you're not sure, say so" in prompts where accuracy matters.

→ 6 more items on the next slide — covering loops, agents, and professional limits.

Pre-flight checklist — continued



Sample tested first

5–10 items processed and cost confirmed before large batch.



Loop has a circuit breaker

Counter or cap stops any API loop if it runs unexpectedly long.



Agent actions reviewed

Human approves before any agent action affects real data or users.



No agent in prod unsupervised

Extensive supervised testing on safe equivalent first.



Time-sensitive info pasted or searched

Not relying on model's training data for current events.



Professional consulted for high-stakes

AI helped me prepare; a licensed expert makes the call.

Full interactive version on the Theory page — tick items off as you set them up. State saves between visits. All 12 checked = green banner = ready to ship safely.

The verify-everything habit

All seven risks in this lesson reduce to a single question to ask before you act on AI output.

"Would I stake something real on this without checking?"

If the answer involves a person's health, money, career, or legal rights — the answer should be no until you verify.

The people who use AI best are not:

- The ones who trust blindly
- The ones who distrust completely
- The ones who never try new things

They are:

- The ones who know the failure modes
- The ones who verify proportionally to stakes
- The ones who built the habits in this lesson

Seven risks. Seven habits. None of them stop you from using AI effectively. All of them keep you safe while you do.

What to do next

Theory page — 4 interactive tools

- **Hallucination Spotting quiz** — identify fabrications in 4 real-style AI responses
- **Privacy Risk Grader** — pick tool + data type, get instant risk verdict
- **Cost Projector** — sliders show cost projection; "runaway" button for a shock
- **Pre-flight Checklist** — complete all 12 items before shipping

Lesson 10 — Glossary & next steps

The final lesson in the AI for Dummies module. A full glossary of every term introduced across lessons 01–09, a cheat-sheet of all key habits, and a curated reading list for going deeper on any topic that caught your interest.

Homework

Three practical tasks that build real instincts: fact-checking an AI response against real sources, running the Privacy Risk Grader on a realistic scenario, and completing the Pre-flight Checklist for a sample project.

Habit to build: Before running any AI automation on real data or real files, run the Pre-flight Checklist. Every time. It becomes second nature within a week — and it has prevented real damage for real people who built the habit.

Safety, privacy, cost, and limitations

Reading time: ~18 min · Prereqs: Lessons 01–08 · 4 interactive tools inside · The owner's manual you wish came with the chatbot.

The thing tutorials skip

Most "getting started with AI" guides show you the impressive parts. The creative writing that writes itself. The code that appears from thin air. The summary that takes seconds instead of hours. What they skip is the part of the owner's manual called *"Cautions and limitations."*

This lesson fills that gap. Seven topics. Each one grounded in something that actually goes wrong for real people. None of them should stop you from using AI — they should help you use it better and safer.

Use the four interactives

This page has four hands-on tools: a Privacy Risk Grader, a Cost Projector, a Hallucination Spotting quiz, and a Pre-flight Checklist. The checklist in particular is meant to be used every time you ship something AI-powered. Complete it before you move on.

1. Hallucinations: confident AI, wrong answer

You've heard the word before. Let's be precise about what it means.

A **hallucination** is when an AI model generates false information and presents it as fact — with complete confidence, fluent prose, and no disclaimer. It's not a bug in the traditional sense. It's a natural consequence of how LLMs work: they predict plausible text, and plausible text is not the same as verified truth.

The most dangerous hallucinations are specific ones. An AI claiming the speed of light is roughly 300,000 km/s is almost certainly right. An AI saying "According to a 2019 paper by Dr. Sarah Chen at MIT titled *Neural Pathway Optimisation in Transformer Architectures...*" is much more likely to be invented, down to the fake author, fake title, and fake institution.

The terrifying part is the tone. The AI sounds exactly the same whether it's right or fabricating. The confidence is baked in. You cannot listen your way to knowing if a claim is hallucinated — you have to check.

Making hallucinations less likely

Add these phrases to prompts where accuracy matters: "*If you're not sure, say so.*" and "*Please indicate your level of confidence.*" This doesn't eliminate hallucinations, but it prompts the model to hedge when it's guessing rather than presenting guesses as certainties. Also: for any specific claim (a statistic, a person's title, a publication), verify it against an independent source before you use it.

INTERACTIVE 1 OF 4 — SPOT THE HALLUCINATION

Each box below is a mock AI response. One sentence in each is the fabricated one. Click the sentence you think is the hallucination. You'll get instant feedback and an explanation.

RESPONSE 1 OF 4 — HISTORY QUESTION

You asked: "Tell me about the Battle of Waterloo."

The Battle of Waterloo was fought on 18 June 1815 near the town of Waterloo in present-day Belgium.

Napoleon's forces were defeated by the Duke of Wellington's coalition army, which included approximately 118,000 troops and was supported by a French naval blockade coordinated from Calais.

The battle effectively ended Napoleon's rule and led to his exile to the island of Saint Helena.

RESPONSE 2 OF 4 — SCIENCE QUESTION

You asked: "What causes rainbows?"

Rainbows form when sunlight enters water droplets in the atmosphere, bends (refracts) as it enters, reflects off the inside back surface, and refracts again as it exits.

The phenomenon was first mathematically described by Ibn al-Haytham in his 1021 work *Kitab al-Manazir*, which included precise measurements of the 42-degree arc angle now called the "al-Haytham constant."

Different wavelengths of light bend at slightly different angles, separating white light into the visible spectrum from red (outermost) to violet (innermost).

RESPONSE 3 OF 4 — TECH QUESTION

You asked: "What is Python used for?"

Python was created by Guido van Rossum and first released in 1991; it is consistently ranked as the most popular programming language in the TIOBE index, which measures language popularity based on a proprietary survey conducted exclusively with Fortune 500 companies.

It is widely used for data science, machine learning, web development, automation, and scientific computing.

Python's design philosophy emphasises code readability, and its syntax allows programmers to express concepts in fewer lines of code than languages like C++ or Java.

RESPONSE 4 OF 4 — HEALTH QUESTION

You asked: "Is coffee good or bad for you?"

Moderate coffee consumption (3–4 cups per day) has been associated in multiple large studies with reduced risk of type 2 diabetes, Parkinson's disease, and liver disease.

A landmark 2018 meta-analysis published in *The Lancet Nutritional Medicine* by Dr. Frank Hu's team at Harvard concluded that the optimal daily dose is precisely 3.7 cups and coined the term "the coffee sweet spot" which is now standard in dietary guidelines.

High caffeine intake can raise blood pressure and disrupt sleep, so people sensitive to caffeine or with certain heart conditions are generally advised to limit their intake.

Score: 0 / 4

Reset quiz

Notice how the fabricated details are always specific and plausible — that's what makes hallucinations dangerous. Vague statements are easy to question; specific false claims are easy to believe.

2. Privacy: what happens to your data

When you type into Claude.ai, ChatGPT, or any cloud AI service, your message travels over the internet to servers you don't control. Depending on the service's terms and your subscription type, that data may be:

- Stored and reviewed by company staff for safety and quality purposes
- Used to improve future model training (often opt-out on paid plans)
- Exposed in a security breach
- Subject to law enforcement requests

None of this means cloud AI is dangerous to use. It means you should think of it the same way you think of sending an email: suitable for most things, not suitable for secrets.

The practical rule: **never paste anything into a public AI chat that you would not put in a company-wide email or a public forum.** This includes passwords, API keys, medical records, financial account details, other people's private information (without their knowledge), and trade secrets.

When privacy is genuinely non-negotiable — you're a healthcare worker handling patient data, a lawyer handling client files, a developer working with production credentials — the answer is a **local model**. Ollama or LM Studio (Module 4) run entirely on your machine. Nothing leaves. The models are somewhat less capable than the top cloud models, but the privacy guarantee is absolute.

INTERACTIVE 2 OF 4 — PRIVACY RISK GRADER

Select the tool you're using and the type of data you're about to paste in. Get an instant verdict.

Tool

— Select tool —

Data type

— Select data type —

This grader encodes common-sense rules. When in doubt, prefer the local model or omit the sensitive detail entirely.

3. Bias: models learned from flawed humans

AI models are trained on text written by people. People hold biases — about gender, race, profession, nationality, and countless other dimensions. The model learns the patterns in that text, which means it absorbs the biases too.

This shows up in subtle ways: occupations assumed to belong to certain genders, certain nationalities associated with certain qualities, advice that reflects the assumptions of the dominant training corpus (which skews heavily English, Western, and internet-connected). More overtly, some models have shown different responses to identical questions framed around different demographic groups.

For casual use, this is worth being aware of but rarely a serious concern. For **decisions that affect people** — hiring, lending, insurance, medical triage — it matters enormously. Using AI to screen CVs at scale, for example, risks encoding and

amplifying whatever biases exist in the training data, applied to thousands of real people's careers.

The practical guidance: add human review and critical thinking wherever AI output touches individual people. Spot-check outputs for patterns. Don't automate at scale what you wouldn't trust on a single case. And know that many jurisdictions are actively legislating around AI in high-stakes decisions: the EU AI Act classifies AI for employment, education, and credit as "high-risk" with mandatory requirements.

Critical: audit before you scale

If you're building any workflow where AI output affects decisions about people, test it manually on 20–50 diverse examples before you automate. Look for patterns. Ask: "Would this output be the same if I changed the name from 'James' to 'Fatima'? From 'London' to 'Lagos'?" If the answer is no and it shouldn't be, you have a bias problem at scale.

4. Cost: tokens add up fast in a loop

Cloud AI APIs charge by the **token**. One token is roughly three-quarters of a word. The cost per token is tiny — fractions of a cent — which makes it easy to underestimate how quickly it accumulates when you're processing large documents, running many calls in a loop, or building an agent that makes requests autonomously.

A common mistake for new developers: writing a loop that processes a large dataset, forgetting to add a counter or cap, and waking up to a large bill. This isn't theoretical. It's a well-documented pattern in developer communities.

The fixes are simple but require doing them proactively:

- **Set a monthly spending limit** in your API console before writing a single line of code that calls an API. Anthropic Console, OpenAI dashboard, and Google AI Studio all have this.
- **Test with a tiny sample first.** Process 5 items. Check the cost. Then extrapolate. Don't run 10,000 items until you've confirmed the per-item cost is acceptable.

- **Use a cheaper model for development.** The smallest model in a family is often good enough for testing and costs a fraction of the flagship.
- **Local models are free per use.** Ollama and LM Studio run on your hardware. The cost is your electricity and the time to set them up once.

INTERACTIVE 3 OF 4 — COST PROJECTOR

Drag the sliders to see how API costs project over time. Then hit "Show runaway agent" for a shock.

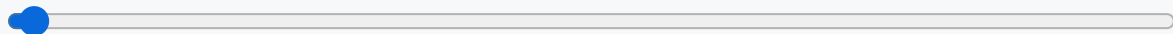
Cost per API call

0.10¢



Calls per day

100



DAILY COST

\$10.00

MONTHLY COST

\$300

YEARLY COST

\$3650

Show me a runaway agent ⚠

⚠ **This is real money.** Set a spending cap in your API console right now. Runaway loops at this scale have caused developers hundreds of dollars in unexpected charges overnight.

5. Agents need a leash

Earlier modules introduced AI agents: tools like Claude Code, AutoGPT, and others that go beyond answering questions to actually taking actions — editing files, running commands, sending requests, browsing the web.

The power here is real, and so is the risk. An agent doesn't have common sense about consequences. If you ask it to "clean up the project folder" and you've given it write access, it might delete things you didn't mean to delete. If you ask it to "send a follow-up email to everyone on the list" and you haven't reviewed the list, it might send 10,000 emails.

The failure mode isn't the agent going "rogue" in a sci-fi sense. It's the agent doing exactly what it understood you to mean — which wasn't quite what you meant. Misunderstood instructions at agent speed cause real damage.

The rules are few but firm:

- **Start on test environments or version-controlled code.** If the agent deletes something on a git repo, you can get it back. If it deletes it on your only copy, you can't.
- **Review before confirming.** Most agent tools show you what they're about to do before executing. Read that. Every time.
- **Scope credentials narrowly.** Don't give an agent your production database password when it only needs to read a config file.
- **Never give unsupervised access to production systems on day one.** Build trust incrementally, the same way you would with a new employee.

Critical: production = supervised

The rule is simple: if an action affects real users, real data, or real money, a human reviews it before it executes. Agent tools are powerful multipliers. The same quality that makes them fast at the right thing makes them fast at the wrong thing. Keep a human in the loop until you've verified the agent's judgement on your specific task through extensive supervised testing.

6. It doesn't know "now"

Every LLM has a **training cutoff**: a date at which its knowledge effectively stops. The model was trained on text collected up to that point, and it has no way to access anything that happened afterwards unless you tell it or a tool retrieves it.

This cutoff is often many months or even more than a year before you're using the model. The most capable models take a long time to train, evaluate, and release. By the time you're chatting with them, their "most recent" knowledge may already be outdated.

The practical implication: **don't ask a standard AI model for time-sensitive information** and trust the result. Current stock prices, election results, today's weather, the latest software versions, recently passed laws — these are all danger zones. The model will either honestly say it doesn't know (good) or fill in a plausible-sounding answer from its training data (bad and potentially outdated).

The workarounds:

- **Paste the current information into your prompt.** If you need the model to reason about a recent news article, copy the article text into your message. The model can analyse information you give it; it just can't fetch it independently.
- **Use a tool with web search enabled.** Claude.ai with search, Perplexity, the web-browsing version of ChatGPT — these retrieve current information before responding. They'll typically mention they're searching. Use them for time-sensitive questions.

7. It's not a professional

This is the limitation that most often comes up in advice-seeking conversations, and the one most people underestimate because the AI's tone is so convincing.

AI is not a doctor, a lawyer, or a financial advisor. It doesn't have a licence. It hasn't examined you. It hasn't read your contract, your medical history, or your tax situation. It knows general patterns from training data. Your situation is specific.

The gap matters most when consequences are high. An AI might describe the typical symptoms of appendicitis correctly but have no way to tell you whether *your* stomach pain is appendicitis, gas, or something else entirely. A lawyer might correctly summarise what a contract clause generally means but not know that in your jurisdiction there's a recent case that changes the interpretation entirely.

The right way to use AI with professional questions: use it to **understand the landscape** and **prepare better questions** for a real professional. "Explain what a non-compete clause is and what I should ask a lawyer about mine" is a great use. "Should I sign this non-compete?" is not one the AI can safely answer.

The "prepare your questions" superpower

AI genuinely shines at turning a scary professional appointment into a productive one. Ask it to explain the terminology you'll encounter, list the questions you should ask your doctor about a diagnosis, or summarise what a financial product is so you can have an informed conversation with an advisor. You'll get more out of the professional meeting, and you'll be a better-informed participant in your own decisions.

Putting it all together: the Pre-flight Checklist

The seven topics above boil down to a set of habits. The checklist below covers everything you should verify before shipping any AI-powered project — a script, an automation, an app, or even a regular workflow. Your state is saved between visits, so you can come back and tick things off as you set them up.

All 12 items must be checked to earn the green banner.

INTERACTIVE 4 OF 4 — PRE-FLIGHT SAFETY CHECKLIST

0 / 12 checked

API KEY & CREDENTIALS SAFETY

- ☐ **API key in environment variable**, not hardcoded in source files or pasted into chat.
- ☐ **Spending limit set** in my API console. I will not be surprised by a large bill from a runaway loop.

PRIVACY REVIEW

- ☐ **No passwords, tokens, or API keys** in any prompt or input that goes to a cloud model.
- ☐ **No unredacted medical, financial, or other people's private data** sent to a cloud service. If this data is involved, I use a local model.

HALLUCINATION VERIFICATION HABIT

- ☐ **I have verified any specific facts, citations, or statistics** the AI produced against an independent source before using them.
- ☐ **My prompts include "If you're not sure, say so"** wherever accuracy matters.

COST & SCALE CONTROLS

- ☐ **I tested with a small sample (5–10 items)** and confirmed cost per item before running any large batch.
- ☐ **Any loop or agent that calls an API has a circuit breaker or counter** that stops it if it runs unexpectedly long.

AGENT & AUTOMATION GUARDRAILS

- ☐ **Agent actions that affect real data or real users are reviewed by a human** before they execute.
- ☐ **I am not running agents against production systems** without extensive supervised testing on a safe equivalent first.

KNOWLEDGE & PROFESSIONAL LIMITS

- ☐ **For time-sensitive questions, I use web search** or paste current info directly — I do not rely on a model's training data for current events.
- ☐ **For medical, legal, or financial decisions, I consult a qualified professional.** AI has helped me prepare; a human expert makes the call.

State is saved in your browser. Come back and check things off as you set them up for a new project.

Summary

Seven risks, seven habits. None of them require you to become an AI safety researcher. They require building a mental checklist you run before trusting output, before pasting

data, before deploying a loop, and before making decisions that affect people.

The goal isn't to make you afraid of AI — it's to make you the kind of person who uses it well for years without getting burned. The people who get the most out of AI tools are not the ones who trust blindly or distrust completely. They're the ones who know the failure modes and work around them.

The homework page gives you three tasks that practice these habits on real scenarios. They're the most important exercises in this course because they build the instincts that protect you in everything that comes after.

Homework — Safety, privacy, cost, and limitations

Estimated time: ~45 min · Tools needed: access to any AI chat (Claude, ChatGPT, etc.) and optionally a search engine · No code required

The seven risks covered in this lesson aren't abstract. They show up in real use. These three tasks give you direct experience with the three most common failure modes — hallucinations, privacy slip-ups, and outdated information — so they become instincts rather than things you have to remember.

Before you start

Have the Theory page open in another tab, particularly the Pre-flight Checklist and the Privacy Risk Grader. You may want to refer back to them during Task 3. Also open Claude.ai, ChatGPT, or any AI chat you use regularly.

Task 1 — Hallucination hunting

This task builds the "verify by default" habit. You're going to deliberately ask AI about things that are easy to get wrong, then check the answers against real sources.

Step A: Choose a domain you know something about. It could be your profession, a hobby, a historical period you've studied, a technical field, or even the city you grew up in. Pick something where you have enough background to recognise a wrong answer.

Step B: Ask the AI five specific factual questions in that domain. Make them specific — not "what is machine learning?" but "What paper first introduced the transformer architecture, who were the authors, and in what year?" Specific claims are where hallucinations hide. Write the AI's answers down (or copy them).

Step C: Verify each answer against an independent source. Use a search engine, Wikipedia, a textbook, or an official source — not another AI. Mark each answer: correct, partially correct, wrong, or unverifiable.

Step D: For any incorrect or suspicious answer, try again with the prompt *"If you're not sure about any part of this, please say so."* Compare the second set of answers.

Reflect and write down:

- Which domain did you choose? Why did you pick it?
- How many of the five answers were fully correct? How many had errors? Were any fabricated?
- Did adding "if you're not sure, say so" change anything? Did the AI flag more uncertainty the second time?
- What type of claim was most likely to be wrong — dates, names, numbers, qualitative descriptions?
- What habit will you apply to your daily AI use based on what you found?

✓ **Self-check: You should have verified five specific AI claims against real sources and identified at least one error or instance of hallucination. If all five were perfect, try a more obscure or niche domain — hallucinations are much more common outside popular knowledge.**

Task 2 — Privacy audit of your own AI use

This task asks you to look back at how you've been using AI and audit it against the privacy rules from this lesson. It's not about catching yourself doing something wrong — it's about building a conscious habit going forward.

Step A: Think back over the last 2–4 weeks of AI use (or longer, if you're new). What kinds of information have you pasted into AI chats? Go back through your actual chat history if you can. Categorise what you've shared:

- Public or general knowledge
- Personal but non-sensitive (your name, city, job title)
- Sensitive personal data (health details, finances, relationships)

- Work or company information
- Other people's information
- Credentials or tokens (hopefully none)

Step B: For each category you've used, use the Privacy Risk Grader in the Theory page to assess whether your tool choice matched the data sensitivity. Were there any mismatches?

Step C: Pick one real task you regularly use AI for — drafting emails, summarising documents, generating code, answering questions — and write a short "privacy rule" for that task. For example: "When I use AI to help draft client emails, I will not include client names, account numbers, or contract details. I will replace them with placeholders like [CLIENT] and [AMOUNT]."

Reflect and write down:

- What data categories have you shared with AI tools? Were any of them higher-risk than you'd realised?
- Were there any mismatches between the sensitivity of your data and the tool you were using?
- Write your personal privacy rule for at least one regular AI task.
- If you handle professionally sensitive data (health, legal, financial, company secrets), do you have a local model set up for those tasks? If not, what would you need to do to set one up?

✓ **Self-check: You should have at least one written privacy rule specific to your own AI use. If your audit turned up no concerns at all, either your use has been very conservative (good) or you haven't looked closely enough (try again with your actual chat history).**

Task 3 — The cutoff test and professional-limits reflection

This task combines the "doesn't know now" and "not a professional" limitations into one practical exercise.

Part A: Testing the knowledge cutoff

Choose a topic where things have changed recently. Good options:

- A software library or framework (e.g., "What's the latest stable version of React, and what did the most recent major release add?")
- A regulatory or legal change in your field or country
- A recent scientific finding or product launch
- Current prices of something (a popular product, an API service you use)

Step A: Ask your AI the question without giving it any current information. Record its answer.

Step B: Search for the current, correct answer from an official source (the library's docs, a government website, a reputable news article).

Step C: Paste the current information into a new prompt and ask the same question, this time providing the real data: *"Here is the current information: [paste text]. Based on this, can you answer [question]?"* Compare the quality of both responses.

Part B: The professional-limits reflection

Think of a real question you've asked AI (or might ask) in one of these areas: health, law, finance, or another professional domain.

Step D: Reframe the question as "help me understand and prepare" rather than "tell me what to do." For example, if the original question was *"Should I take this medication?"*, reframe it as *"What are the standard considerations a doctor weighs when prescribing this medication? What questions should I ask my doctor at my appointment?"*

Step E: Compare the original question (if you ran it) with the reframed version. What's different about the output?

Reflect and write down:

- What topic did you choose for the cutoff test? How out of date was the AI's unaided answer?
- Did providing the current information in the prompt improve the response? How?
- What is your rule for when to use a search-enabled tool vs. a standard AI chat?

- What professional question did you choose for Part B? How did reframing it change the nature of the answer?
- How will you use AI for professional-domain questions going forward?

✓ **Self-check:** You should have one concrete example of the knowledge cutoff causing a wrong or outdated answer, and one example of reframing a professional question from "tell me what to do" to "help me understand and prepare." Both should feel immediately applicable to how you use AI.

What good looks like after this lesson

After completing these three tasks, you should be doing three things automatically: verifying specific AI claims before acting on them, thinking about data sensitivity before pasting anything into a chat, and reaching for a search-enabled tool or a real professional whenever currency or expertise is non-negotiable. These three habits will protect you from the vast majority of AI-related mistakes.

Lesson 10: Glossary, cheat sheet, next steps

Lesson 10 — Glossary, cheat sheet & next steps

You made it. Here's the map for going further.

AI for Dummies

Final Lesson

Reference

Wrap-up

Ten lessons ago you didn't know what an LLM was. Today you can install a local model, call an API, and build a working bot. This slide deck is your permanent reference — vocabulary locked in, cheat sheet packed, road ahead mapped.

Questions you can now answer

Before this course, these were mysteries. Now they're old news.

✓ What's the difference between AI, ML, deep learning, and an LLM?

✓ Why does AI make things up — and what's a hallucination?

✓ What is a token, and why does it matter for cost?

✓ What is a context window and what happens when you exceed it?

✓ How do I run a model on my own computer with no data leaving?

✓ What is an API key and how do I keep it safe?

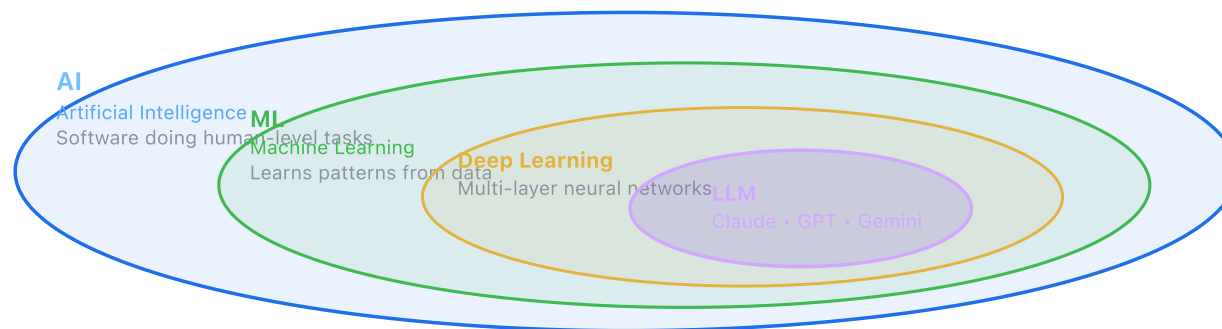
✓ How do I write a Python script that calls an AI?

✓ What is RAG, and when do I need it?

✓ What is an agent — and what makes it different from a chat?

✓ What is AGI — and does it exist yet?

Glossary spotlight — core concepts



CORE

AI

Software doing tasks that normally need human intelligence — language, images, planning, prediction.

CORE

Machine Learning

AI that learns patterns from data instead of hand-written rules.

CORE

Deep Learning

ML using multi-layer neural networks. Responsible for most AI progress since 2012.

CORE

LLM

Large Language Model: a text-trained model like Claude, ChatGPT, or Gemini. Predicts the next token.

Glossary spotlight — working with AI

The four terms that show up in every single AI conversation.

WORK

Prompt

What you type to the AI.
The whole craft of getting good results is learning to write better prompts.
Recipe: Role + Context + Task + Format.

WORK

Token

A chunk of text — roughly $\frac{3}{4}$ of a word. AI reads and charges by the token, not words or characters. "Hello world" $\approx$ 3 tokens.

WORK

Context window

How much text the model can consider at once. Past the limit, the model "forgets" the beginning of your conversation.

RISK

Hallucination

Confident, false AI output stated as fact, with the same fluent tone as correct answers. You cannot tell from tone alone. **Always verify what matters.**

Remember: confident tone $\neq$ accurate. The model sounds equally sure whether it is right or completely wrong. Verify anything consequential.

Glossary spotlight — doorways & tools

TOOL

API

Application Programming Interface — the way your programs send prompts to and get answers from an AI service over the internet.

TOOL

CLI / Terminal

The text window where you type commands. Used to run Ollama, Python scripts, and Claude Code. Looks intimidating; it's just typing.

TOOL

Local model

A model running on your own computer. No data leaves your machine. Free, private, offline-capable.

WORK

Agent

AI that takes multi-step actions toward a goal — can call tools, browse the web, or run code. Claude Code is an agent.

WORK

RAG

Retrieval-Augmented Generation — feeding the AI snippets of your own documents so it answers from them instead of guessing.

CORE

Multimodal

A model that handles more than text — also images, audio, video, or documents. Most modern frontier models are multimodal.

Glossary spotlight — the bigger picture

BIG PICTURE

AGI

Artificial General Intelligence — hypothetical human-level general AI that can do anything a human can. **Does not exist yet.** Much debated. Don't confuse it with today's impressive-but-narrow systems.

BIG PICTURE

Parameters

A model's internal knobs — “7B” means 7 billion parameters. More parameters generally means smarter but heavier and slower.

BIG PICTURE

Generative AI

AI that creates new content: text, images, audio, video, or code. LLMs are one type; image generators (Stable Diffusion, DALL·E) are another.

Narrow AI is all that exists today — good at specific tasks. AGI (human-level general capability) is a future hypothesis, not a current product.

Model sizes: 3B = phone-friendly • 7B = laptop • 70B = needs a powerful GPU • >100B = datacenter. Start small, scale up as needed.

One-page cheat sheet — part 1

To just use AI

- Go to **claude.ai** and type. That's it.
- Free tier is real — you can do useful work without spending anything.
- Also: ChatGPT (chatgpt.com) • Gemini (gemini.google.com)

To get good answers

- **Recipe:** Role + Context + Task + Format
- Show an example output if you have one
- If the first answer is wrong, say what was wrong and ask again
- Be specific — vague question = vague answer

Example prompt using the recipe: "You are a friendly lawyer [role]. I have a dispute with my landlord about a broken boiler [context]. Explain what rights I have under UK law [task]. Use bullet points, plain English, no jargon [format]."

One-page cheat sheet — part 2

To run AI privately on your computer

- **Easy route (GUI):** Download LM Studio, search for a model, download, run
- **Terminal route:** Install Ollama, then:

```
ollama pull llama3.2:3b
```

```
ollama run llama3.2:3b
```
- Nothing leaves your computer. No API key. No costs.

To let AI write & edit code

- Install Claude Code (requires Node.js 18+):

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

```
bash
```
- Navigate to your project folder and run `claude`
- **Alternative:** Cursor (cursor.sh) — VS Code fork with AI built in

Hardware check: 8 GB RAM runs small models; 16 GB is comfortable; 32 GB+ for the big ones. Apple Silicon Macs are excellent for local AI.

One-page cheat sheet — part 3

To automate with code

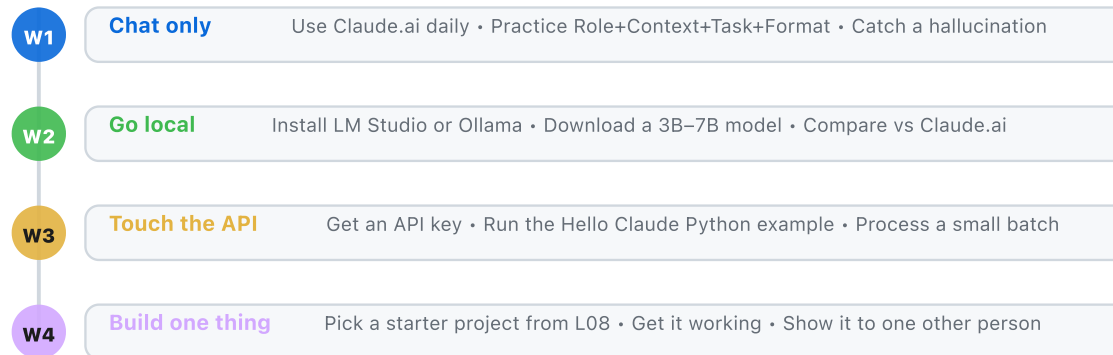
- Get an API key at **console.anthropic.com**
- Store it in a `.env` file: `ANTHROPIC_API_KEY=sk-ant-...`
- Install the SDK: `pip install anthropic python-dotenv`
- Start with a cheap model (`claude-haiku-4-5`) & set `max_tokens`
- Only upgrade to a larger model once logic is confirmed working

Always (the three rules)

- **Verify what matters.** AI is a confident guesser, not a fact database.
- **Protect private data.** Don't paste passwords or confidential records into a cloud chat.
- **Keep a human in the loop** for anything medical, legal, financial, or safety-critical.

The full interactive cheat sheet (expandable • printable • 6 sections) lives on the Theory page. Hit Expand all then ⌘P to print a one-pager.

Your 4-week path to confident AI use



This is a practice plan, not a course. Each week adds one capability. The Theory page has a full checkbox tracker — state saved in your browser.

Week 1 spotlight — Chat only

Week
1

Goal: build the habit of actually using AI every day

- Use Claude.ai for at least one real task every day (writing, research, summarising)
- Write a prompt using the full recipe: Role + Context + Task + Format
- Take a mediocre answer and iterate on it until it's good
- Catch the AI making something up — verify with a real source and note the discrepancy
- Explain what an LLM is to someone who doesn't know — without jargon

Don't move on until: you feel comfortable iterating on an answer with follow-up messages, and you've caught at least one hallucination.

Tip: Use it for something you actually care about — a real email, a real question, a real problem at work. Fake exercises don't build habits.

Week 2 spotlight — Go local

Week
2

Goal: run AI on your own machine with no internet

- Install LM Studio (lmstudio.ai) or Ollama (ollama.com)
- Download a small local model (3B–7B parameters)
- Ask the same question to both your local model and Claude.ai — compare quality
- Turn off WiFi and confirm your local model still works

Quick-start commands (Ollama):

```
ollama pull llama3.2:3b  
ollama run llama3.2:3b
```

First run downloads the model (~2 GB). After that it loads in seconds.

Week 3 spotlight — Touch the API

Week
3

Goal: make your first automated AI call from Python

- Get an Anthropic API key at console.anthropic.com
- Store the key in a `.env` file — never in your code
- Run the Hello Claude Python example from Lesson 6
- Process a small batch with a loop (e.g., summarise 5 items)
- Check your usage console — understand what you spent

```
# The whole thing in 8 lines
import anthropic
client = anthropic.Anthropic()           # reads ANTHROPIC_API_KEY from env
msg = client.messages.create(
    model="claude-haiku-4-5",
    max_tokens=256,
    messages=[{"role": "user", "content": "Say hello in one sentence."}],
```

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

Week 4 spotlight — Build one thing

Week
4

Goal: ship something real, however small

- Pick one starter project from Lesson 8 (or write your own idea)
- Get a "hello world" working — even if it's ugly
- Make it do the one thing you actually wanted it to do
- Show it to one other person (even if they aren't technical)

Starter project ideas

- Summarise all your email newsletters in one daily digest
- AI that answers questions about your own notes / documents
- Script that rewrites your job descriptions in plain language

The mindset shift: You're not trying to build the perfect thing. You're trying to go from zero-to-shipped once — because the second project is always easier than the first.

- CLI tool that translates files in bulk
-

Where to go next — curated links

[Anthropic Prompting Guide](#) ↗

Official techniques, examples, and patterns for prompting Claude.

[Anthropic API Docs](#) ↗

Complete reference for the Anthropic API — endpoints, SDKs, examples.

[Ollama](#) ↗

Download & run local models via terminal. One command per model.

[LM Studio](#) ↗

Desktop app for local AI models — no terminal, friendly GUI.

[Hugging Face](#) ↗

The GitHub of AI models. Browse thousands, run demos in-browser.

[Andrej Karpathy \(YouTube\)](#) ↗

Deep but accessible explanations of how neural networks & LLMs work.

[Cursor](#) ↗

VS Code fork with AI built in — great if you prefer an editor over a terminal tool.

[fast.ai](#) ↗

Free practical deep learning course — best path from beginner to practitioner.

The one rule — reprised

//

AI is confidently wrong sometimes.

Verify what matters.

What “verify” looks like: if the AI tells you a legal deadline, a drug interaction, a financial figure, or any claim you’re going to act on — check it against a primary source. The AI is not that source.

What it does NOT mean: you don’t need to fact-check every sentence. Use judgment. Verify the things that matter; let the draft writing slide.

This habit — more than any technical skill — is what separates a confident, responsible AI user from someone who gets burned. Build it in Week 1 and never lose it.

You finished AI for Dummies

//

You don't need to understand how an engine works to drive a car. The same is true here. Start with Chat, build the verify-everything habit, and let curiosity pull you toward the next thing. That's it.

Lesson 10 complete

Course complete

You're dangerous now

Go build something

Next step: open the [Theory page](#), bookmark it, and start the 4-week tracker.

Then: open [Homework](#), pick your final project, and write your AI elevator pitch.

Glossary, cheat sheet, and next steps

Your permanent reference for the whole course — plus a clear path forward.

You've reached the final lesson. This page is designed to live in your bookmarks. It has the full glossary you can search, an interactive cheat sheet you can print, a 4-week action plan with checkboxes, and a curated list of where to go next.

How to use this page: Work through the content once. Then bookmark it. Come back when you need to look something up, print the cheat sheet, or check off your weekly progress.

The full glossary

Twenty-plus terms you've met across this course. Search by name or definition, or filter by category.

All

Concepts

Tools

Risks

Clear

AI	Software doing tasks that normally need human intelligence — understanding language, recognising images, answering questions, writing, planning, and predicting.	Concepts
Machine Learning (ML)	AI that learns patterns from data instead of hand-written rules. "Here are 10 million emails — figure out what spam looks like."	Concepts
Deep Learning	ML that uses multi-layer neural networks. Responsible for most of the progress in AI since ~2012.	Concepts

LLM

Large Language Model — a text-trained model like Claude, ChatGPT, or Gemini. Predicts the next token based on everything it has read.

[Concepts](#)**Generative AI**

AI that creates new content — text, images, audio, video, or code. LLMs are one type; image generators (Stable Diffusion, DALL·E) are another.

[Concepts](#)**Model**

The trained AI itself — the swappable "engine." Claude Opus, Llama 3, Gemini Flash are all different models.

[Concepts](#)**Parameters**

A model's internal knobs — "7B" means 7 billion parameters. More parameters generally means smarter but also heavier and slower.

[Concepts](#)**Prompt**

What you type to the AI. The whole craft of getting good results is learning to write better prompts.

[Concepts](#)**Token**

A chunk of text — roughly $\frac{3}{4}$ of a word. AI reads and charges by the token, not by words or characters.

[Concepts](#)**Context window**

How much text the model can consider at once. Past the limit, the model "forgets" the beginning of your conversation.

[Concepts](#)**Multimodal**

A model that handles more than text — also images, audio, video, or documents. Most modern frontier models are multimodal.

[Concepts](#)**Agent**

AI that takes multi-step actions toward a goal rather than just answering once. It can call tools, browse the web, or run code.

[Concepts](#)

RAG

Retrieval-Augmented Generation — feeding the AI relevant snippets of your own documents so it can answer from them instead of guessing.

Concepts**AGI**

Artificial General Intelligence — hypothetical human-level general AI that can do anything a human can. Does not exist yet.

Concepts**API**

Application Programming Interface — the way your programs send prompts to and get answers from an AI service over the internet.

Tools**API key**

Your secret password for using an API. Anyone who gets it can bill to your account. Store it in a `.env` file and never commit it to git.

Tools**CLI / terminal**

The text window where you type commands directly to your computer. Used to run Ollama, Python scripts, and Claude Code.

Tools**Local model**

A model running entirely on your own computer. No data leaves your machine — useful for privacy, offline work, or just experimenting for free.

Tools**Ollama**

A command-line tool for downloading and running local models. One command to download a model, one to run it.

Tools**LM Studio**

A desktop app (Mac/Windows/Linux) for running local AI models with a friendly GUI — no terminal required.

Tools**Claude Code**

Anthropic's AI coding tool that runs in your terminal and can read, write, and edit files in your project. Great for code generation and refactoring.

Tools**Hallucination**

Confident, false AI output — stated as fact, with the same fluent tone as correct answers. You cannot tell from tone alone. Always verify what matters.

Risks

Bias

AI inheriting unfair patterns from its training data — over-representing some groups, under-representing others, or reflecting historical discrimination.

Risks

Prompt injection

An attack where malicious text in input data tricks the AI into following hidden instructions — bypassing your intended instructions.

Risks

One-page cheat sheet

Everything you actually need to remember, compressed into six cards. Click any card to expand it. Hit **Expand all** then **⌘P** to print a one-pager.

To just use AI

Go to **claude.ai** and type. That's it.

Free tier is real — you can do useful work without spending anything. Create an account with a Google or email login and start a conversation.

Other options: **ChatGPT** (chatgpt.com), **Gemini** (gemini.google.com). They all work similarly from the chat interface.

To get good answers

The recipe: Role + Context + Task + Format.

"You are a friendly lawyer. I have a dispute with a landlord about a broken boiler [context]. Explain what rights I have under UK law [task]. Use bullet points, plain English, no jargon [format]."

Tips:

- Show an example output if you have one.
- If the first answer is wrong, say what was wrong and ask again — don't start over.
- Be specific. Vague question = vague answer.

To run AI privately on your computer

Easy route (GUI): Download [LM Studio](#), open it, search for a model (try `Llama-3.2-3B` for speed, `Mistral-7B` for quality), download, run.

Terminal route: Install [Ollama](#), then:

```
ollama pull llama3.2:3b
```

```
ollama run llama3.2:3b
```

Nothing leaves your computer. No API key. No costs.

To let AI write & edit code

Install Claude Code (requires Node.js 18+):

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

Then navigate to your project folder and run `claude`. It reads your files and can edit them directly.

Alternative: **Cursor** ([cursor.sh](#)) — a VS Code fork with AI built in; good for beginners who prefer a traditional editor.

To automate with code

1. Get an API key at [console.anthropic.com](#).
2. Store it in a `.env` file: `ANTHROPIC_API_KEY=sk-ant-...`
3. Install the SDK: `pip install anthropic python-dotenv`
4. Start with a cheap model (`claude-haiku-4-5`) and set `max_tokens` to cap costs.
5. Only upgrade to a larger model when you've confirmed the logic works.

Always (the three rules)

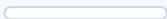
- 1. Verify what matters.** AI is a confident guesser, not a fact database. Check anything consequential against a real source.
- 2. Protect private data.** Don't paste passwords, patient records, or confidential company data into a cloud chat box.

3. Keep a human in the loop. For anything with real-world consequences — medical, legal, financial, safety-critical — a human makes the final call.

A suggested 4-week path

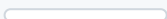
This is not a course. It's a practice plan. Each week adds one new capability. Check off tasks as you go — your progress is saved in your browser.

Week 1 — Chat only

 0%

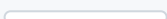
- ☐ Use Claude.ai every day this week for at least one real task (writing, research, summarising)
- ☐ Write a prompt using the full recipe (Role + Context + Task + Format)
- ☐ Take a mediocre answer and iterate on it with a follow-up until it's good
- ☐ Catch the AI making something up — verify with a real source and note the discrepancy
- ☐ Explain what an LLM is to someone who doesn't know — without jargon

Week 2 — Go local

 0%

- ☐ Install LM Studio or Ollama on your computer
- ☐ Download a small local model (3B–7B parameters)
- ☐ Ask the same question to both your local model and Claude.ai — compare quality
- ☐ Turn off your WiFi and confirm your local model still works

Week 3 — Touch the API

 0%

- ☐ Get an Anthropic API key at console.anthropic.com
- ☐ Store the key in a .env file (never in your code)
- ☐ Run the "Hello, Claude" Python example from Lesson 6
- ☐ Process a small batch of something with a loop (e.g., summarise 5 items from a list)
- ☐ Check your Anthropic usage console and understand what you spent

Week 4 — Build one tiny thing

 0%

- ☐ Pick a starter project from Lesson 8 (or write your own idea)
- ☐ Get a "hello world" working — even if it's ugly
- ☐ Make it do the one thing you actually wanted it to do
- ☐ Show it to one other person (even if they aren't technical)

Reset all progress

Where to go next

A curated selection of resources, grouped by what you want to do. All open in a new tab.

PROMPTING & BEST PRACTICES

[Anthropic Prompting Guide](#)



The official guide to prompt engineering with Claude — techniques, examples, and patterns.

[Prompt Engineering Guide](#)



Community-maintained comprehensive guide covering CoT, few-shot, RAG, and more.

[Learn Prompting](#)



Free, beginner-friendly course on prompt engineering — no code required.

API & SDKS

[Anthropic API Docs](#)



Complete reference for the Anthropic API — endpoints, parameters, SDKs, and examples.

[Claude Code Docs](#)



How to install and use Claude Code — the AI coding tool that lives in your terminal.

[Cursor](#)



VS Code fork with AI built in — great if you prefer a traditional editor over a terminal tool.

LOCAL AI

[Ollama](#)



The easiest way to download and run local models via the terminal. One command per model.

[LM Studio](#)



Desktop app for running local AI models — no terminal needed, friendly GUI.

[AnythingLLM](#)



Run a local RAG system — chat with your own documents using a local model. No data leaves your computer.

COMMUNITIES & LEARNING

[r/LocalLLaMA](#)



The best community for local AI news — model releases, benchmarks, and tips.

[Hugging Face](#)



The GitHub of AI models and datasets. Browse thousands of open models, run demos in your browser.

[Andrej Karpathy \(YouTube\)](#)



Deep but accessible explanations of how neural networks and LLMs actually work — from an OpenAI co-founder.

[fast.ai](#)



Free practical deep learning course — the best way to go from beginner to practitioner.

The whole secret: You don't need to understand how an engine works to drive a car. The same is true here. Start with Chat, build the verify-everything habit, and let curiosity pull you toward the next thing. That's it.

Homework — Glossary, cheat sheet, and next steps

The final homework. Three reflective tasks and one mini-project suggestion.

This is the final lesson of AI for Dummies. The homework here is intentionally reflective rather than technical. There's no code to write and no solution to check. The goal is to consolidate what you know and make a concrete decision about what to do next.

Task 1 — The elevator pitch

Imagine you're in a lift with someone who just heard the word "AI" on the news and asks you: *"Wait, what actually **is** AI? Is it really going to take everyone's jobs?"*

Write a 60-second spoken answer. No jargon. No hedging. No "well, it depends." Be honest, be concrete, and end with one thing they should do this week to see it for themselves.

Tips for a good answer:

- Start with what AI is in one sentence (not what it can do).
- Give one concrete example they've definitely encountered (spam filters, phone autocomplete, Netflix recommendations).
- Acknowledge the job question honestly — don't dismiss it, don't catastrophise.
- End with an action: "Go to claude.ai and ask it to explain your next meeting agenda."

Self-check: Read your answer out loud. Does it take under 60 seconds? Would a non-technical friend nod along or glaze over? If the latter, cut more jargon.

Task 2 — Your personal AI audit

Look at the last week of your life — work, personal, creative, or otherwise. Identify **three tasks** where you used AI, tried to use AI, or could have used AI but didn't.

For each task, answer:

1. What was the task?
2. Did AI help, fail, or get in the way? Why?
3. What would you do differently next time?

One of the three should be a case where AI made things worse — gave you wrong information, a generic output, or something you had to redo entirely. That's not a failure. That's calibration. The goal is to know where AI is worth using and where it's not.

Self-check: Are your "next time" answers specific? "I'll write a better prompt" is not specific. "I'll add the format constraint and a worked example" is specific.

Task 3 — Pick one thing and commit

Look at the 4-week path on the Theory page. Look at your current checkbox state. Identify the **single next action** you haven't done yet and write it down as a commitment with a specific date:

"By [date], I will [specific action]."

Examples of specific commitments:

- "By Friday, I will install Ollama and download the Llama 3.2 3B model."
- "By next Tuesday, I will write a Python script that calls the Claude API and summarises my last five emails."
- "By Sunday, I will finish the chatbot from Lesson 8 and show it to my partner."

The rule: if you can't finish the action in under two hours, it's too big. Break it in half.

Self-check: Does your commitment have a specific date and a specific, observable outcome? "Get better at AI" is not a commitment. "Run the Hello World example" is.

Mini-project idea — Build your own "AI for Dummies" explainer

This isn't a task to submit — it's an invitation. The best way to test whether you really understand something is to try to teach it.

Here's the idea:

Make a one-page "AI starter guide" for someone in your life who knows nothing about it. It could be a parent, a younger sibling, a colleague from a non-technical department, or a neighbour. Tailor it to them specifically.

It could be:

- A single printed page (use the cheat sheet from Theory as a starting point).
- A short voice memo where you explain the top five things they need to know.
- A 10-minute screen share where you walk them through claude.ai live.
- A Google Doc you write together.

The constraint: it must be genuinely useful to *that specific person*, not to you. If it works for you but they still look confused, it doesn't count.

Why this matters: teaching forces you to confront every spot where you sort-of understand something. You'll find three or four gaps you didn't know you had. That's the point.

You finished AI for Dummies. Ten lessons, from "what even is an LLM" to local models, the API, building a bot, and thinking about safety. You have the

vocabulary, the habits, and the tools to go wherever curiosity takes you next. The Theory page has the glossary, cheat sheet, and links whenever you need them. Now go build something that didn't exist before you started this course.