

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

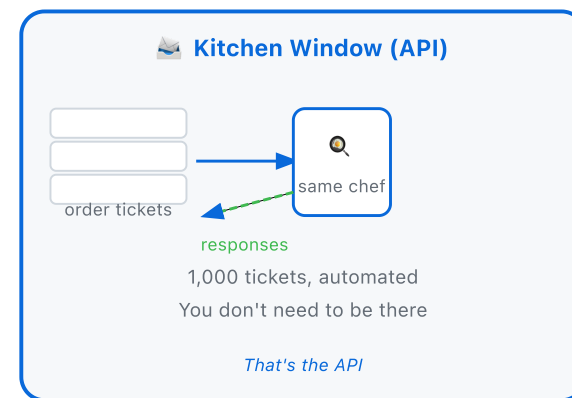
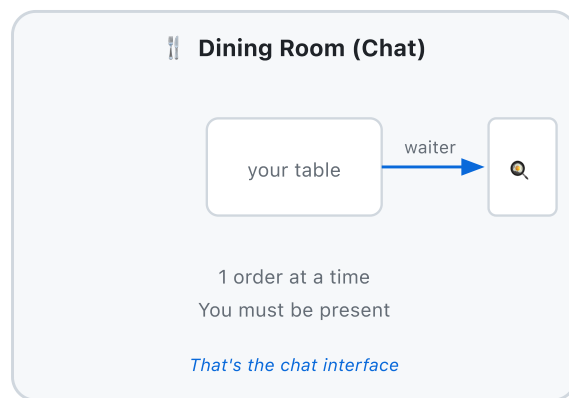
Your program sends prompts. It collects replies.
You're not even at the keyboard.

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

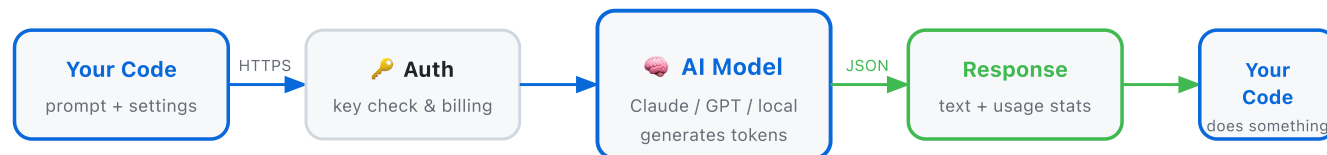
Rule of thumb: if you'd paste the same prompt more than a handful of times, the API is the right door.
Learn it once; use it everywhere.

The kitchen window — same chef, different door



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

Anatomy of an API call



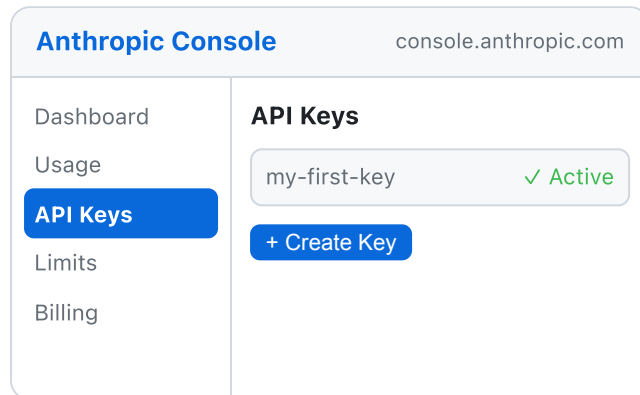
Round trip: typically 1–3 seconds for a short reply

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.

