

lab

April 3, 2026

1 Lesson 02 Lab — Claude API Setup & First Calls

In this lab you will make your first real API calls to Claude, implement streaming, and build a multi-turn conversation loop.

Work through each exercise in order. Replace every `raise NotImplementedError(...)` with your implementation.

1.1 Prerequisites

Before running this notebook:

1. Install dependencies: `pip install anthropic python-dotenv`
2. Create a `.env` file in the same directory as this notebook with:
`ANTHROPIC_API_KEY=sk-ant-api03-your-key-here`
3. Get your key from console.anthropic.com → API Keys
4. Add `.env` to your `.gitignore` — never commit your real key!

```
[ ]: # Setup cell - run this first
from dotenv import load_dotenv
import os
import anthropic

load_dotenv() # loads ANTHROPIC_API_KEY from .env into os.environ

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

print("Setup complete. Client type:", type(client))
print("API key loaded:", bool(os.environ.get("ANTHROPIC_API_KEY")))
```

1.2 Exercise 1 — Your First API Call

Use `client.messages.create()` to make a basic (non-streaming) API call.

Your task: - Call the API with `model="claude-3-5-haiku-20241022"` and `max_tokens=256` - Use the message: "Tell me a fun fact about large language models." - Print the response text - Print the token usage (input and output tokens)

Expected output:

Response:

<some fun fact about LLMs>

Token usage:

Input tokens: 17

Output tokens: ~50

Stop reason: end_turn

```
[ ]: # TODO: Make your first API call
# Use model="claude-3-5-haiku-20241022", max_tokens=256
# Message: "Tell me a fun fact about large language models."
# Print: the response text AND the usage stats
raise NotImplementedError("Complete the TODO above")
```

1.3 Exercise 2 — Streaming Responses

Implement `stream_response(prompt)` that uses the streaming API to print text as it arrives.

Your task: - Use `client.messages.stream()` as a context manager - Iterate over `stream.text_stream` and print each chunk with `end=""` and `flush=True` - After the loop, print a newline - Test it with the prompt: "Explain quantum entanglement in simple terms."

Expected behaviour: Text should appear incrementally, not all at once.

```
[ ]: def stream_response(prompt: str) -> None:
    """Stream a response from Claude, printing each chunk as it arrives."""
    # TODO: Use client.messages.stream() context manager
    # Iterate stream.text_stream and print each chunk with end="" and flush=True
    raise NotImplementedError("Implement streaming")

# Test it:
stream_response("Explain quantum entanglement in simple terms.")
```

1.4 Exercise 3 — Multi-turn Conversation

Implement `chat_loop()` — an interactive chatbot that remembers the full conversation history.

Your task: - Maintain a history list of `{"role": ..., "content": ...}` dicts - On each turn: append the user message to history, call the API with the full history, append the assistant response, print it - Exit the loop when the user types "quit" - Test that Claude remembers your name across turns

Test conversation:

You: My name is Alex.

Claude: Nice to meet you, Alex! ...

You: What is my name?
Claude: Your name is Alex!
You: quit

```
[ ]: def chat_loop() -> None:
    """Run an interactive multi-turn conversation with Claude."""
    history = []
    print("Chat with Claude (type 'quit' to exit)\n")
    while True:
        user_input = input("You: ").strip()
        if user_input.lower() == "quit":
            break
        # TODO: append user message to history, call API, append assistant
        ↪ response, print
        raise NotImplementedError("Implement the chat loop body")

chat_loop()
```