

lab__answer

April 3, 2026

1 Lab 08 — RAG Basics (ANSWERS)

Build an end-to-end RAG pipeline with ChromaDB, sentence-transformers, and Claude.

Exercises: 1. Embed and store 10 documents in ChromaDB 2. Implement `rag_answer()` — retrieve + generate 3. Metadata filtering comparison

```
[ ]: # Setup
import os
from pathlib import Path
from dotenv import load_dotenv
import anthropic
import chromadb
from sentence_transformers import SentenceTransformer

load_dotenv()
client = anthropic.Anthropic()
MODEL = "claude-sonnet-4-20250514"
embed_model = SentenceTransformer("all-MiniLM-L6-v2")
chroma_client = chromadb.Client()
print("Setup complete.")
```

1.1 Exercise 1 — Embed and Store Documents

```
[ ]: # Load all 10 sample documents
docs_dir = Path("demo/sample_docs")
doc_files = sorted(docs_dir.glob("*.txt"))

doc_ids = []
doc_texts = []
doc_metadatas = []

for i, fpath in enumerate(doc_files):
    text = fpath.read_text(encoding="utf-8")
    doc_id = fpath.stem # e.g., "doc_01_pathlib"
    # Docs 1-5 get topic="python", docs 6-10 get topic="stdlib"
    topic = "python" if i < 5 else "stdlib"
```

```

doc_ids.append(doc_id)
doc_texts.append(text)
doc_metadatas.append({"topic": topic, "filename": fpath.name})

print(f"Loaded {len(doc_texts)} documents")

# Embed all documents
embeddings = embed_model.encode(doc_texts).tolist()
print(f"Generated {len(embeddings)} embeddings of dimension_
↳ {len(embeddings[0])}")

# Store in ChromaDB
collection = chroma_client.create_collection("python_docs")
collection.add(
    ids=doc_ids,
    documents=doc_texts,
    embeddings=embeddings,
    metadatas=doc_metadatas,
)
print(f"Collection '{collection.name}' has {collection.count()} documents")

```

```

[ ]: # Test Exercise 1
results = collection.query(query_texts=["file path handling"], n_results=3)
print("Top 3 results for 'file path handling':")
for doc in results["documents"][0]:
    print(f" - {doc[:80]}...")

```

1.2 Exercise 2 — RAG Answer Function

```

[ ]: def rag_answer(question: str, collection, top_k: int = 3) -> str:
    """Retrieve top_k docs from collection, build prompt, get Claude's answer.
    ↳ """

    # 1. Embed the question
    query_emb = embed_model.encode([question]).tolist()

    # 2. Retrieve top_k documents
    results = collection.query(query_embeddings=query_emb, n_results=top_k)
    retrieved_docs = results["documents"][0]

    # 3. Build context string
    context = "\n\n---\n\n".join(retrieved_docs)

    # 4. Build prompt and send to Claude
    prompt = f"""Use the following context to answer the question.
    If the context does not contain enough information, say so.
    """

```

```

Context:
{context}

Question: {question}"""

    response = client.messages.create(
        model=MODEL,
        max_tokens=1024,
        messages=[{"role": "user", "content": prompt}],
    )
    return response.content[0].text

```

```

[ ]: # Test Exercise 2
print(rag_answer("How do I read a JSON file in Python?", collection))
print("----")
print(rag_answer("What is a defaultdict?", collection))

```

1.3 Exercise 3 — Metadata Filtering

```

[ ]: # Filtered query: only topic="python" (docs 1-5)
filtered = collection.query(
    query_texts=["iteration patterns"],
    n_results=3,
    where={"topic": "python"}
)
print("Filtered (topic=python):")
for doc, meta in zip(filtered["documents"][0], filtered["metadatas"][0]):
    print(f"    [{meta['filename']}] {doc[:80]}...")

print()

# Unfiltered query
unfiltered = collection.query(
    query_texts=["iteration patterns"],
    n_results=3
)
print("Unfiltered:")
for doc, meta in zip(unfiltered["documents"][0], unfiltered["metadatas"][0]):
    print(f"    [{meta['filename']}] {doc[:80]}...")

# Analysis: filtering narrows results to the python-tagged subset,
# which may or may not improve relevance depending on the query.
# For "iteration patterns", itertools (doc_03, topic=python) should rank high
# in both cases, but filtering excludes stdlib docs that might also be relevant.

```