```
"""
Assignment 2: Personalized Course Recommendation Engine (Strict PDF Version)
-----------------------------------------------------------------------------
- Uses Azure OpenAI embeddings
- Vector DB: FAISS (in-memory)
- Loads courses from assignment2dataset.csv
- Returns top 5 most relevant courses by cosine similarity
- Uses only provided 5 test profiles from assignment2 PDF
- Prints recommendations and brief comment on each
"""

import os
import pandas as pd
from typing import List, Tuple
from langchain_openai import AzureOpenAIEmbeddings
from langchain.vectorstores import FAISS
from langchain.schema import Document

# === ENVIRONMENT VARIABLES ===
AZURE_OPENAI_ENDPOINT = os.environ["AZURE_OPENAI_ENDPOINT"]
AZURE_OPENAI_API_KEY = os.environ["AZURE_OPENAI_API_KEY"]

# === Load and Prepare Course Data ===
df = pd.read_csv("assignment2dataset.csv")
assert "course_id" in df.columns
assert "description" in df.columns

# === Setup Azure OpenAI Embeddings ===
embeddings = AzureOpenAIEmbeddings(
    azure_endpoint=AZURE_OPENAI_ENDPOINT,
    api_key=AZURE_OPENAI_API_KEY,
    model="text-embedding-ada-002",
    api_version="2023-05-15"
)

# === Embed and Index All Courses with FAISS ===
docs = df["description"].tolist()
course_ids = df["course_id"].astype(str).tolist()
metas = df.drop(columns=["description"]).to_dict(orient="records")
documents = [Document(page_content=desc, metadata=meta) for desc, meta in zip(docs, metas)]

print("Embedding and indexing all courses (first run may take a minute)...")
vectorstore = FAISS.from_documents(documents, embeddings)
print(f"Indexed {len(course_ids)} courses.")

# === Assignment-Compliant Recommendation Function ===
def recommend_courses(profile: str, completed_ids: List[str]) -> List[Tuple[str, float]]:
    """
    Returns a list of (course_id, similarity_score) for the top-5 recommendations.
    """
    # Per assignment, combine profile and completed_ids for the embedding query
    query_str = profile.strip()
    if completed_ids:
        query_str += " Completed: " + ", ".join(str(cid) for cid in completed_ids)
    results = vectorstore.similarity_search_with_score(query_str, k=5)
    output = []
    for doc, score in results:
        output.append((
            doc.metadata.get("course_id", "unknown"),
            float(1 - score)  # Convert distance to similarity (cosine)
        ))
    return output

# === EXACT 5 Sample Queries ==
sample_cases = [
    {
        "profile": "IвЂ™ve completed the вЂPython Programming for Data SciencevвЂ™ course and enjoy data visualization. What should I take next?",
        "completed_ids": []  # Not specified, treat as empty list
    },
    {
        "profile": "I know Azure basics and want to manage containers and build CI/CD pipelines. Recommend courses.",
        "completed_ids": []
    },
    {
        "profile": "My background is in ML fundamentals; IвЂ™d like to specialize in neural networks and production workflows.",
        "completed_ids": []
    },
    {
        "profile": "I want to learn to build and deploy microservices with KubernetesвЂ”what courses fit best?",
        "completed_ids": []
    },
    {
        "profile": "IвЂ™m interested in blockchain and smart contracts but have no prior experience. Which courses do you suggest?",
        "completed_ids": []
    }
]

if __name__ == "__main__":
    print("\n==== Assignment 2: Test Results and Comments ====\n")
    for idx, case in enumerate(sample_cases, 1):
        print(f"--- Test Profile {idx} ---")
        print("Profile:", case["profile"])
        print("Completed:", case["completed_ids"])
        results = recommend_courses(case["profile"], case["completed_ids"])
        for rank, (cid, score) in enumerate(results, 1):
            print(f"Rank {rank}: {cid} (Similarity: {score:.3f})")
        print("Comment: These recommendations are generated purely by semantic similarity to the test profile, using Azure OpenAI embeddings. The ranking should reflec
```