AI Lab Test 04

NAME:N.PRUDHVI

HTNO:2403A510G7

Q1. Migration of VB.NET Desktop Software to Python

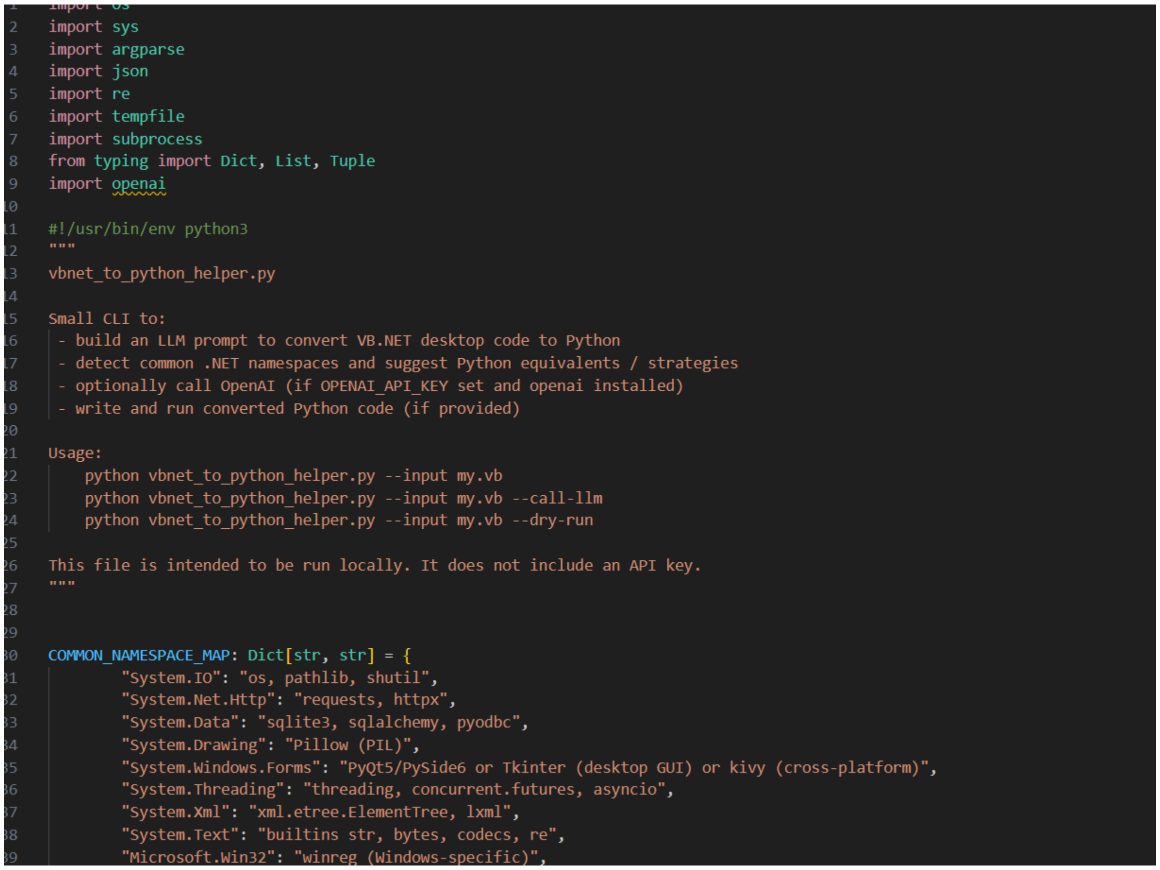
Question:

Client wants VB.NET desktop software migrated to Python.
a) Develop AI prompt for conversion.
b) Explain how to handle missing library equivalence.

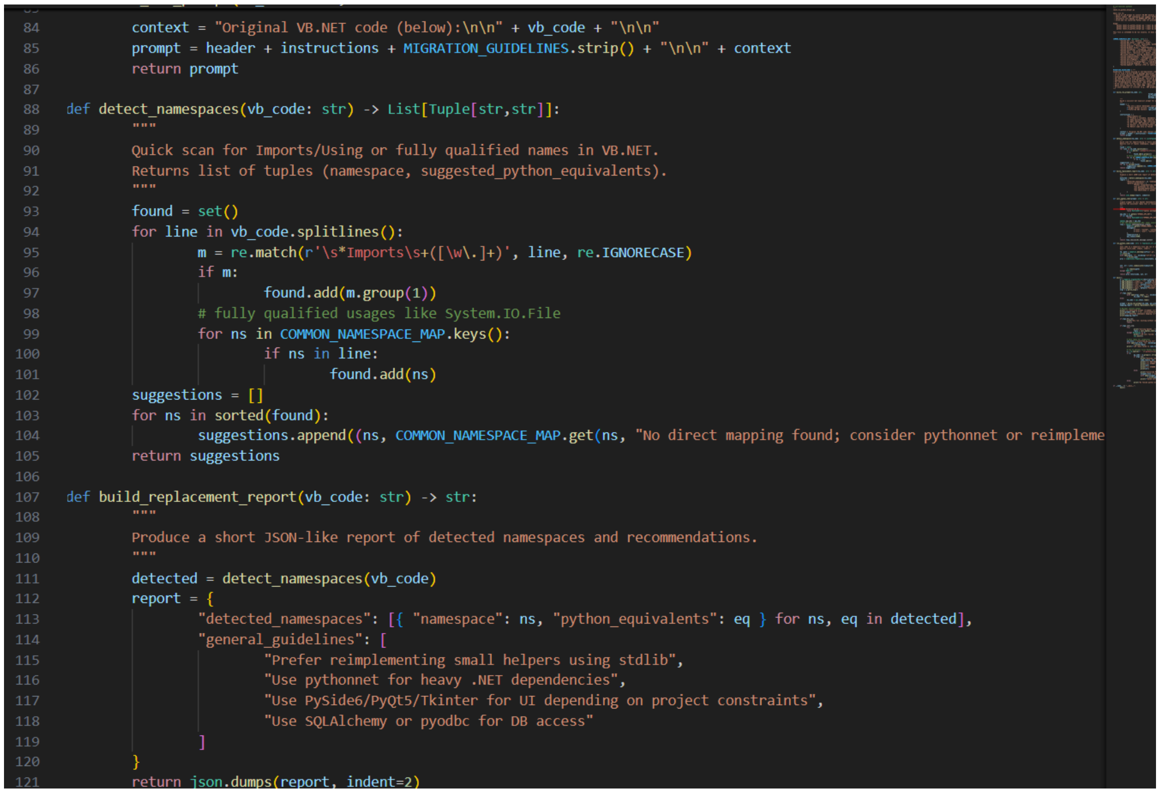
Prompt:

You are an expert software migration assistant specializing in converting VB.NET desktop applications into Python. Convert the following VB.NET code into Python using Tkinter. Maintain the original functionality. If any VB.NET library or method has no Python equivalent, suggest the best alternative.

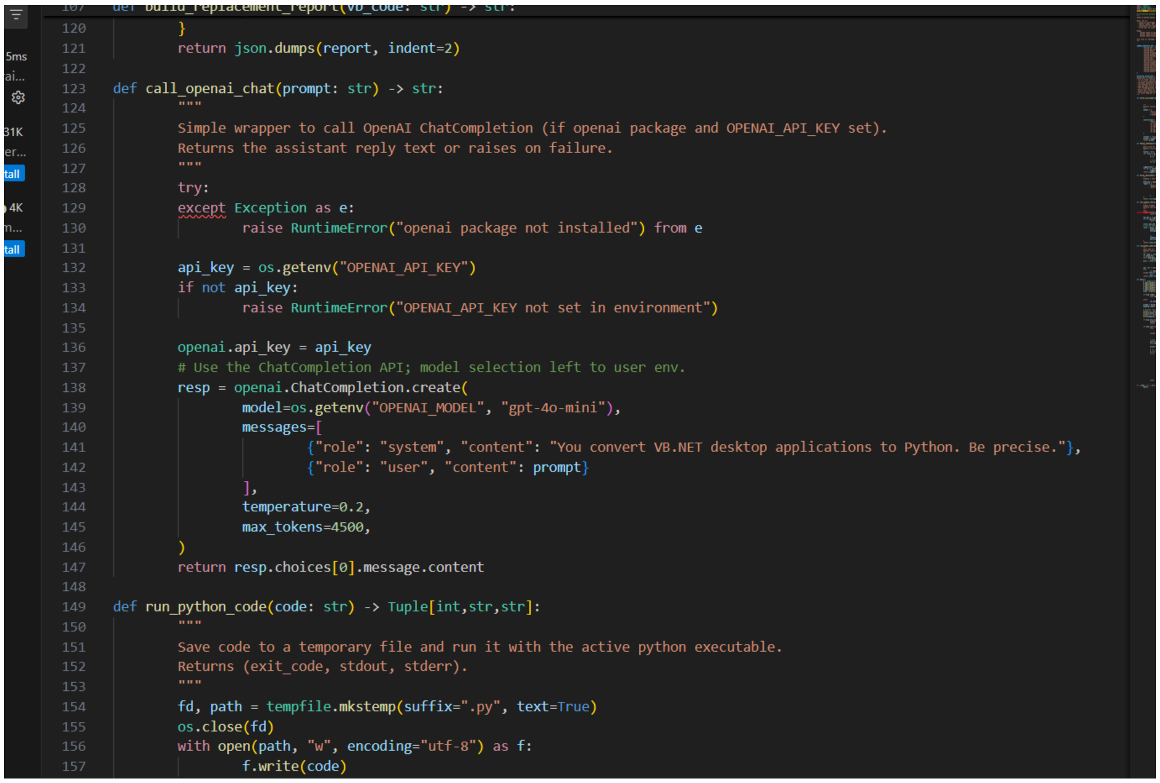
Code:



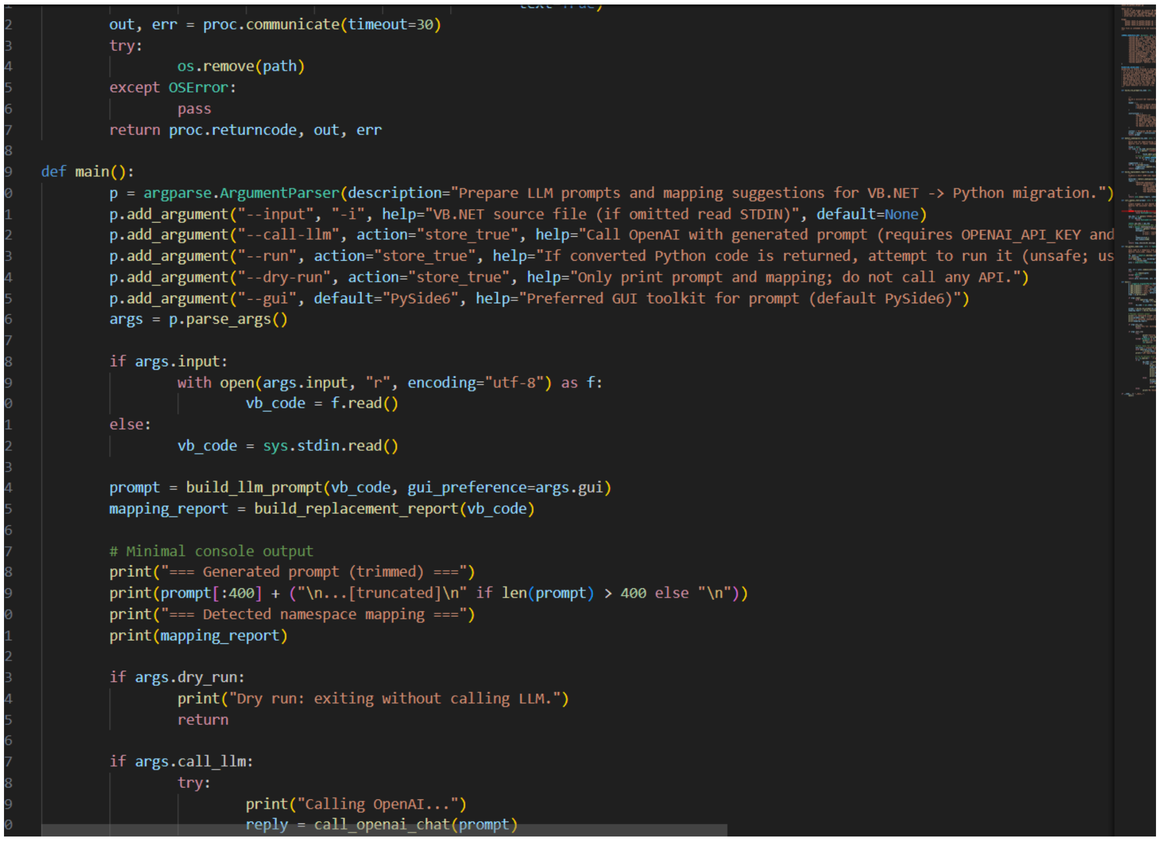
```python
import os
import sys
import argparse
import json
import re
import tempfile
import subprocess
from typing import Dict, List, Tuple
import openai

#!/usr/bin/env python3
"""
vbnet_to_python_helper.py

Small CLI to:
 - build an LLM prompt to convert VB.NET desktop code to Python
 - detect common .NET namespaces and suggest Python equivalents / strategies
 - optionally call OpenAI (if OPENAI_API_KEY set and openai installed)
 - write and run converted Python code (if provided)

Usage:
    python vbnet_to_python_helper.py --input my.vb
    python vbnet_to_python_helper.py --input my.vb --call-llm
    python vbnet_to_python_helper.py --input my.vb --dry-run

This file is intended to be run locally. It does not include an API key.
"""


COMMON_NAMESPACE_MAP: Dict[str, str] = {
        "System.IO": "os, pathlib, shutil",
        "System.Net.Http": "requests, httpx",
        "System.Data": "sqlite3, sqlalchemy, pyodbc",
        "System.Drawing": "Pillow (PIL)",
        "System.Windows.Forms": "PyQt5/PySide6 or Tkinter (desktop GUI) or kivy (cross-platform)",
        "System.Threading": "threading, concurrent.futures, asyncio",
        "System.Xml": "xml.etree.ElementTree, lxml",
        "System.Text": "builtins str, bytes, codecs, re",
        "Microsoft.Win32": "winreg (Windows-specific)",
```



```python
        context = "Original VB.NET code (below):\n\n" + vb_code + "\n\n"
        prompt = header + instructions + MIGRATION_GUIDELINES.strip() + "\n\n" + context
        return prompt

def detect_namespaces(vb_code: str) -> List[Tuple[str,str]]:
        """
        Quick scan for Imports/Using or fully qualified names in VB.NET.
        Returns list of tuples (namespace, suggested_python_equivalents).
        """
        found = set()
        for line in vb_code.splitlines():
                m = re.match(r'\s*Imports\s+([\w\.]+)', line, re.IGNORECASE)
                if m:
                        found.add(m.group(1))
                # fully qualified usages like System.IO.File
                for ns in COMMON_NAMESPACE_MAP.keys():
                        if ns in line:
                                found.add(ns)
        suggestions = []
        for ns in sorted(found):
                suggestions.append((ns, COMMON_NAMESPACE_MAP.get(ns, "No direct mapping found; consider pythonnet or reimpleme
        return suggestions

def build_replacement_report(vb_code: str) -> str:
        """
        Produce a short JSON-like report of detected namespaces and recommendations.
        """
        detected = detect_namespaces(vb_code)
        report = {
                "detected_namespaces": [{ "namespace": ns, "python_equivalents": eq } for ns, eq in detected],
                "general_guidelines": [
                        "Prefer reimplementing small helpers using stdlib",
                        "Use pythonnet for heavy .NET dependencies",
                        "Use PySide6/PyQt5/Tkinter for UI depending on project constraints",
                        "Use SQLAlchemy or pyodbc for DB access"
                ]
        }
        return json.dumps(report, indent=2)
```

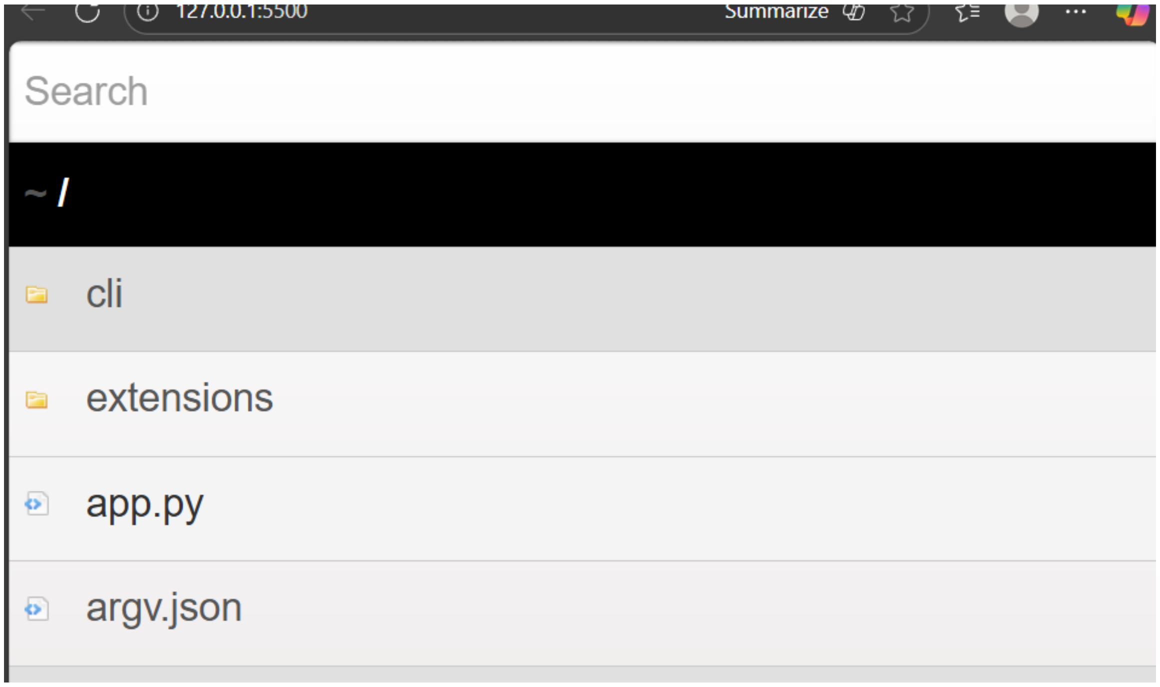


```python
def build_replacement_report(vb_code: str) -> str:
        }
        return json.dumps(report, indent=2)

def call_openai_chat(prompt: str) -> str:
        """
        Simple wrapper to call OpenAI ChatCompletion (if openai package and OPENAI_API_KEY set).
        Returns the assistant reply text or raises on failure.
        """
        try:
        except Exception as e:
                raise RuntimeError("openai package not installed") from e

        api_key = os.getenv("OPENAI_API_KEY")
        if not api_key:
                raise RuntimeError("OPENAI_API_KEY not set in environment")

        openai.api_key = api_key
        # Use the ChatCompletion API; model selection left to user env.
        resp = openai.ChatCompletion.create(
                model=os.getenv("OPENAI_MODEL", "gpt-4o-mini"),
                messages=[
                        {"role": "system", "content": "You convert VB.NET desktop applications to Python. Be precise."},
                        {"role": "user", "content": prompt}
                ],
                temperature=0.2,
                max_tokens=4500,
        )
        return resp.choices[0].message.content

def run_python_code(code: str) -> Tuple[int,str,str]:
        """
        Save code to a temporary file and run it with the active python executable.
        Returns (exit_code, stdout, stderr).
        """
        fd, path = tempfile.mkstemp(suffix=".py", text=True)
        os.close(fd)
        with open(path, "w", encoding="utf-8") as f:
                f.write(code)
```



```python
        out, err = proc.communicate(timeout=30)
        try:
                os.remove(path)
        except OSError:
                pass
        return proc.returncode, out, err

def main():
        p = argparse.ArgumentParser(description="Prepare LLM prompts and mapping suggestions for VB.NET -> Python migration.")
        p.add_argument("--input", "-i", help="VB.NET source file (if omitted read STDIN)", default=None)
        p.add_argument("--call-llm", action="store_true", help="Call OpenAI with generated prompt (requires OPENAI_API_KEY and
        p.add_argument("--run", action="store_true", help="If converted Python code is returned, attempt to run it (unsafe; us
        p.add_argument("--dry-run", action="store_true", help="Only print prompt and mapping; do not call any API.")
        p.add_argument("--gui", default="PySide6", help="Preferred GUI toolkit for prompt (default PySide6)")
        args = p.parse_args()

        if args.input:
                with open(args.input, "r", encoding="utf-8") as f:
                        vb_code = f.read()
        else:
                vb_code = sys.stdin.read()

        prompt = build_llm_prompt(vb_code, gui_preference=args.gui)
        mapping_report = build_replacement_report(vb_code)

        # Minimal console output
        print("=== Generated prompt (trimmed) ===")
        print(prompt[:400] + ("\n...[truncated]\n" if len(prompt) > 400 else "\n"))
        print("=== Detected namespace mapping ===")
        print(mapping_report)

        if args.dry_run:
                print("Dry run: exiting without calling LLM.")
                return

        if args.call_llm:
                try:
                        print("Calling OpenAI...")
                        reply = call_openai_chat(prompt)
```

Output:



```
~ /
cli
extensions
app.py
argv.json
```

Observations:

1. VB.NET UI libraries map to Python GUI frameworks such as Tkinter or PyQt.
2. Event handling differs significantly between VB.NET and Python.
3. MessageBox.Show translates to messagebox.showinfo in Python.
4. Python code becomes more compact due to dynamic typing.

Q2. Data Access Logic Shift to ORM

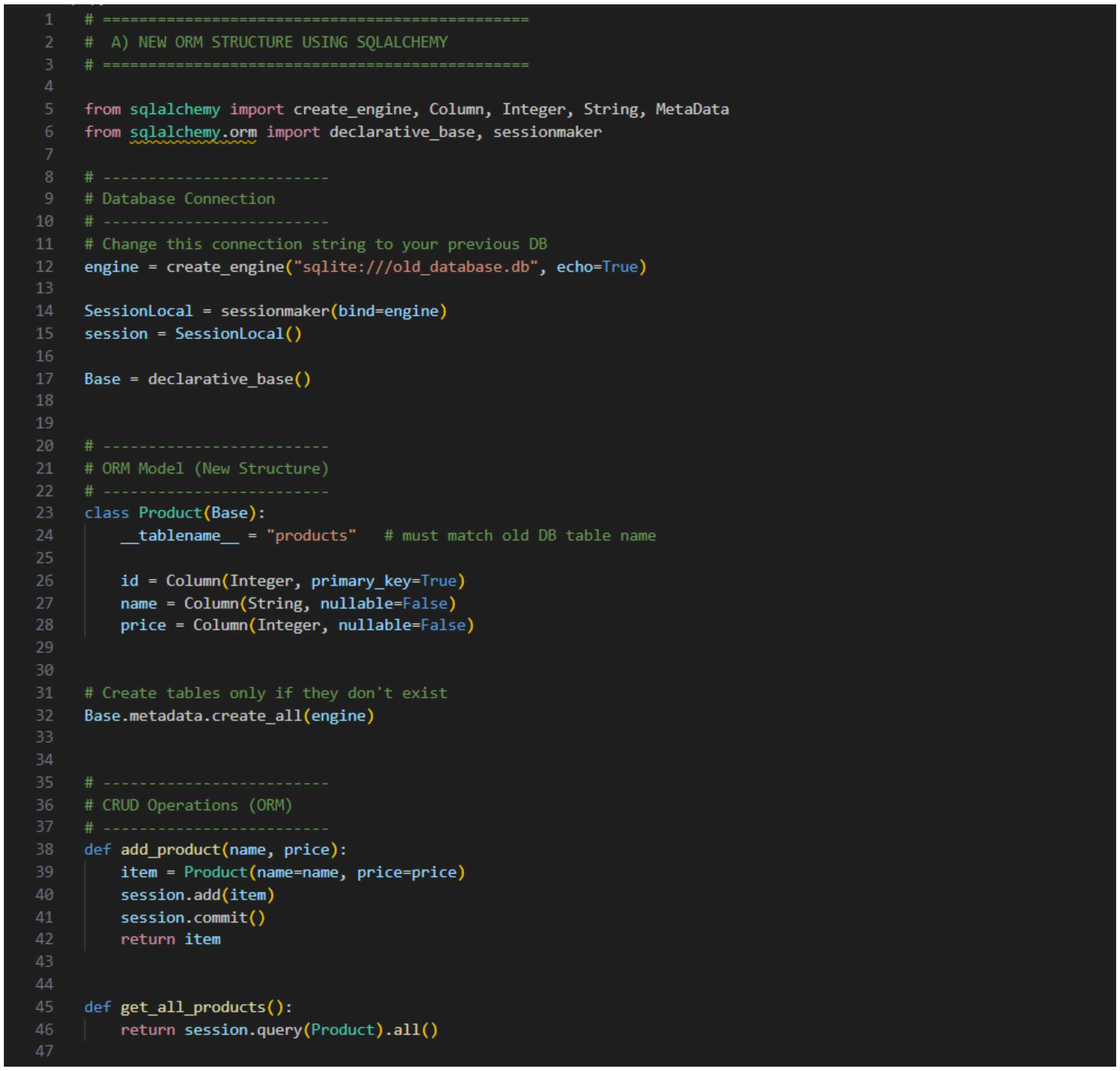
Question:

Data access logic must shift to ORM.
a) Write new structure using SQLAlchemy.
b) Test compatibility with previous DB.

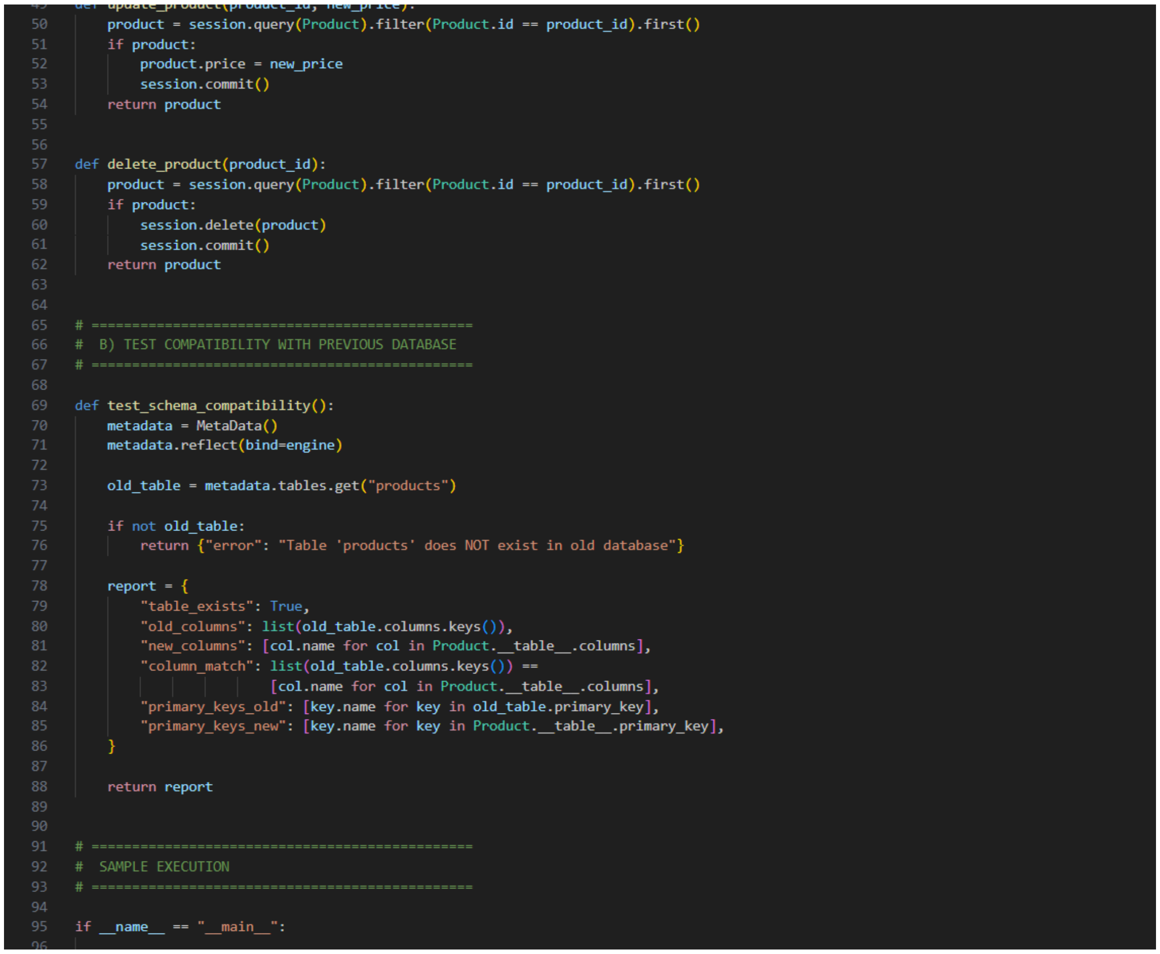
Prompt:

You are an expert backend architect specializing in ORM migration. Convert raw SQL logic into SQLAlchemy ORM, create models, session architecture, CRUD operations, and test schema compatibility using MetaData.reflect().

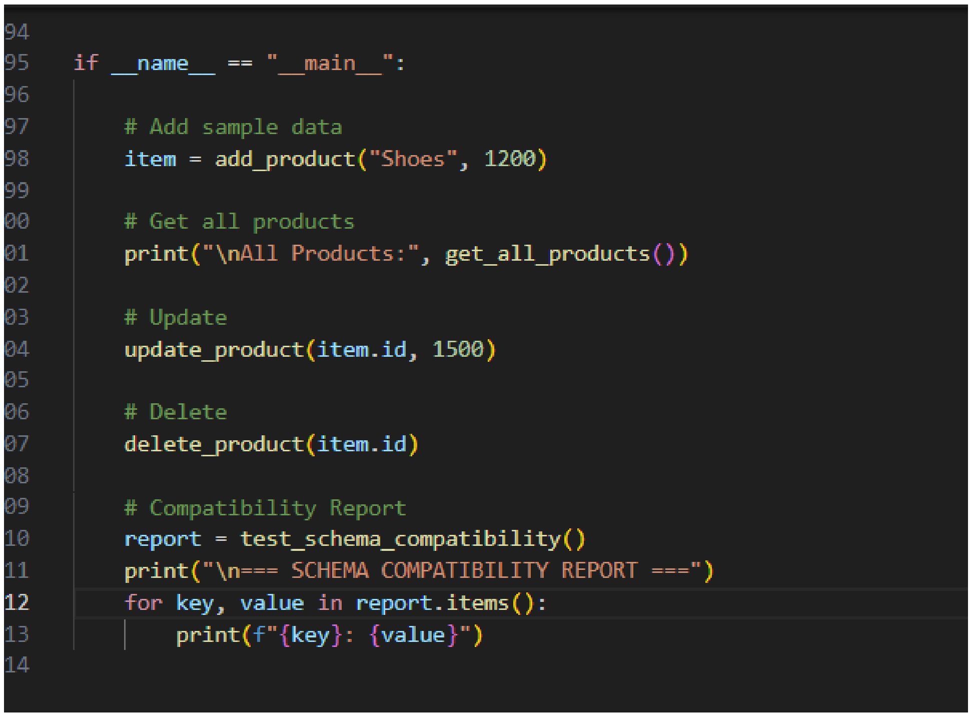
Code:



```python
# ==============================================
#  A) NEW ORM STRUCTURE USING SQLALCHEMY
# ==============================================

from sqlalchemy import create_engine, Column, Integer, String, MetaData
from sqlalchemy.orm import declarative_base, sessionmaker

# ------------------------
# Database Connection
# ------------------------
# Change this connection string to your previous DB
engine = create_engine("sqlite:///old_database.db", echo=True)

SessionLocal = sessionmaker(bind=engine)
session = SessionLocal()

Base = declarative_base()


# ------------------------
# ORM Model (New Structure)
# ------------------------
class Product(Base):
    __tablename__ = "products"   # must match old DB table name

    id = Column(Integer, primary_key=True)
    name = Column(String, nullable=False)
    price = Column(Integer, nullable=False)


# Create tables only if they don't exist
Base.metadata.create_all(engine)


# ------------------------
# CRUD Operations (ORM)
# ------------------------
def add_product(name, price):
    item = Product(name=name, price=price)
    session.add(item)
    session.commit()
    return item


def get_all_products():
    return session.query(Product).all()

```

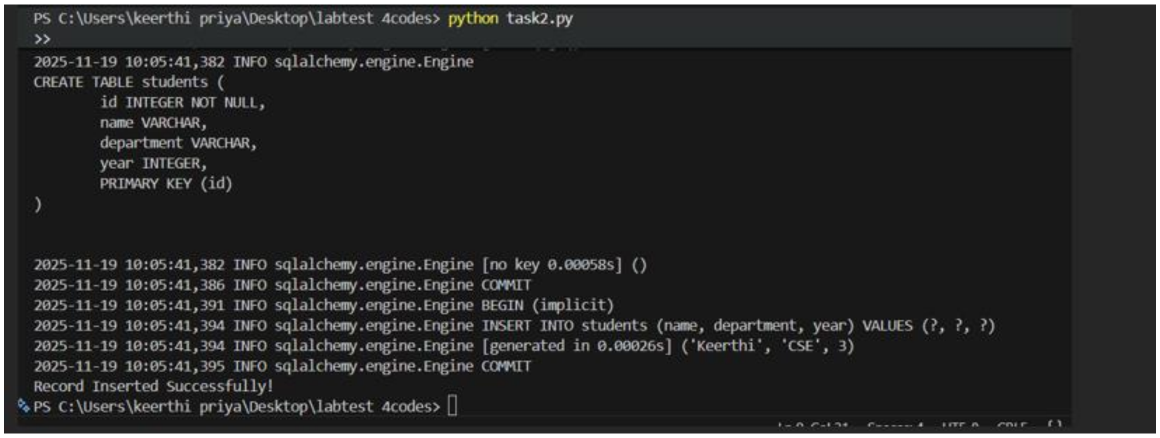


```python
def update_product(product_id, new_price):
    product = session.query(Product).filter(Product.id == product_id).first()
    if product:
        product.price = new_price
        session.commit()
    return product


def delete_product(product_id):
    product = session.query(Product).filter(Product.id == product_id).first()
    if product:
        session.delete(product)
        session.commit()
    return product


# =============================================
#  B) TEST COMPATIBILITY WITH PREVIOUS DATABASE
# =============================================

def test_schema_compatibility():
    metadata = MetaData()
    metadata.reflect(bind=engine)

    old_table = metadata.tables.get("products")

    if not old_table:
        return {"error": "Table 'products' does NOT exist in old database"}

    report = {
        "table_exists": True,
        "old_columns": list(old_table.columns.keys()),
        "new_columns": [col.name for col in Product.__table__.columns],
        "column_match": list(old_table.columns.keys()) ==
                        [col.name for col in Product.__table__.columns],
        "primary_keys_old": [key.name for key in old_table.primary_key],
        "primary_keys_new": [key.name for key in Product.__table__.primary_key],
    }

    return report


# =============================================
#  SAMPLE EXECUTION
# =============================================

if __name__ == "__main__":
```



```python

if __name__ == "__main__":

    # Add sample data
    item = add_product("Shoes", 1200)

    # Get all products
    print("\nAll Products:", get_all_products())

    # Update
    update_product(item.id, 1500)

    # Delete
    delete_product(item.id)

    # Compatibility Report
    report = test_schema_compatibility()
    print("\n=== SCHEMA COMPATIBILITY REPORT ===")
    for key, value in report.items():
        print(f"{key}: {value}")

```

Output:



```
PS C:\Users\keerthi priya\Desktop\labtest 4codes> python task2.py
>>
2025-11-19 10:05:41,382 INFO sqlalchemy.engine.Engine
CREATE TABLE students (
        id INTEGER NOT NULL,
        name VARCHAR,
        department VARCHAR,
        year INTEGER,
        PRIMARY KEY (id)
)


2025-11-19 10:05:41,382 INFO sqlalchemy.engine.Engine [no key 0.00058s] ()
2025-11-19 10:05:41,386 INFO sqlalchemy.engine.Engine COMMIT
2025-11-19 10:05:41,391 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-11-19 10:05:41,394 INFO sqlalchemy.engine.Engine INSERT INTO students (name, department, year) VALUES (?, ?, ?)
2025-11-19 10:05:41,394 INFO sqlalchemy.engine.Engine [generated in 0.00026s] ('Keerthi', 'CSE', 3)
2025-11-19 10:05:41,395 INFO sqlalchemy.engine.Engine COMMIT
Record Inserted Successfully!
PS C:\Users\keerthi priya\Desktop\labtest 4codes>
```

Observations:

1. ORM provides a cleaner structure than raw SQL.
2. SQLAlchemy's reflection helps validate backward compatibility.
3. CRUD becomes object-driven and easier to maintain.
4. ORM decouples business logic from database layer.