AI Homework Jun 3

*Objective: Learn the **Assistants API** (with built-in `file_search`) and **Structured Output** by building a mini AI tutor that pulls answers from uploaded study materials and then produces concise revision notes.*

Repo scaffold

```
study-assistant-lab/
│
├─ README.md
├─ requirements.txt          # openai>=1.83.0, python-dotenv, pydantic
├─ .env.example              # OPENAI_API_KEY
│
├─ data/                      # place your PDF(s) here
│   └─ calculus_basics.pdf
│
├─ scripts/
│   ├─ 00_bootstrap.py        # create/reuse assistant with file_search
│   ├─ 01_qna_assistant.py    # Part 1 solution
│   ├─ 02_generate_notes.py   # Part 2 solution
│   └─ 99_cleanup.py
│
└─ tests/
    └─ test_notes_schema.py   # optional pytest validation
```

Part 1 — Q&A Assistant from PDFs (≈ 60 min)

> **Goal:** Build an assistant that answers study questions by retrieving passages from the uploaded PDF(s).

1. **Bootstrap the assistant** (run `00_bootstrap.py`). For example:

```python
assistant = client.assistants.create(
    name="Study Q&A Assistant",
    instructions=(
        "You are a helpful tutor. "
```

```
        "Use the knowledge in the attached files to answer questions.
"
        "Cite sources where possible."
    ),
    model="gpt-4o-mini",
    tools=[{"type": "file_search"}]
)
```

2. **Upload one or more course PDFs**. For example:

```
file_id = client.files.create(
    purpose="knowledge_retrieval",
    file=open("data/calculus_basics.pdf", "rb")
).id
client.assistants.update(
    assistant.id,
    tool_resources={"file_search": {"vector_store_files": [file_id]}}
)
```

3. **Interact via threads** (`01_qna_assistant.py`)
   - Prompt examples to test:
     - “Explain the difference between a definite and an indefinite integral in one paragraph.”
     - “Give me the statement of the Mean Value Theorem.”
   - Stream the run; print both answer and the `citations` field to verify retrieval.
4. **Self-check**: answer references at least one chunk ID from the PDF.

Part 2 — Generate 10 Exam Notes (≈ 45 min)

> **Goal:** Produce exactly **ten** bite-sized revision notes in JSON, enforcing a schema with Structured Output.

1. **Define a Pydantic schema**

```
from pydantic import BaseModel, Field

class Note(BaseModel):
    id: int                = Field(..., ge=1, le=10)
    heading: str           = Field(..., example="Mean Value Theorem")
    summary: str           = Field(..., max_length=150)
    page_ref: int | None   = Field(None, description="Page number in
source PDF")
```

2. **Prompt in JSON-mode** (`02_generate_notes.py`)

```
system = (
    "You are a study summarizer. "
    "Return exactly 10 unique notes that will help prepare for the
exam. "
    "Respond *only* with valid JSON matching the Note[] schema."
)

response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[{"role": "system", "content": system}],
    response_format={"type": "json_object"}
)
data = json.loads(response.choices[0].message.content)
notes = [Note(**item) for item in data["notes"]]  # will raise if
invalid
```

3. **Print pretty notes** or save to `exam_notes.json` .