🎓 Classroom Exercise: Building a Single-Agent LangGraph App with Internal Looping

🌟 Learning Objectives

By completing this exercise, you will be able to:

1. Define and manage agent **state** in LangGraph
2. Build a **single-agent graph** that processes multiple tasks autonomously
3. Implement **internal recursion** using conditional edges
4. Use **one `graph.invoke()`** to drive the entire agentic application
5. Configure LangGraph's **max recursion limit**
6. Integrate file-based prompt input

🧩 Exercise Overview

In this exercise, you will build a LangGraph application with:

- One agent node
- A state object containing multiple prompts
- A loop that runs ***inside* the agent**, not in Python
- A single **`graph.invoke()` call**
- Recursion enabled up to 100 cycles

At runtime, the agent should:

1. Read a list of prompts from `prompts.txt`
2. Process each prompt one by one
3. Store the outputs in the agent state
4. Stop automatically when all prompts are handled

Your entire multi-prompt workflow must execute with **one single invocation** of the graph.

📝 Starter Code (You Will Complete & Run)

Provide this starter code skeleton to students, with TODO markers where they must fill in logic.

---

exercise_agent_loop.py

```python
"""
Classroom Exercise: Single-Agent LangGraph Loop

Goal:
- Build a single agent that processes a list of prompts
  using one graph.invoke() call.
- The agent must loop internally using conditional edges.
- Max recursion limit: 100
"""

from langgraph.graph import StateGraph, END
from langchain_openai import ChatOpenAI


# ----------------------------------------------------------
# 1. TODO: Define the AgentState
# ----------------------------------------------------------

class AgentState(dict):
    """
    State fields:
      - prompts: list[str]
      - index: int (current prompt)
      - outputs: list[str]

    TODO: Fill in the type hints above (optional)
    """
    pass


# ----------------------------------------------------------
# 2. TODO: Implement the agent node
# ----------------------------------------------------------

llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)

def agent_node(state: AgentState):
    """
    Steps:
    - Read current index
    - If finished, return state unchanged
    - Otherwise, run LLM on prompts[index]
    - Append result to outputs
    - Increment index
    - Return new state
    """

    # TODO: Extract fields from state

    # TODO: Check if index >= len(prompts)

    # TODO: Run the LLM on the current prompt

    # TODO: Append output & increment index

    # TODO: Return updated state
    pass


# ----------------------------------------------------------
# 3. TODO: Add conditional loop
# ----------------------------------------------------------
```

```
def should_continue(state: AgentState):
    """
    If all prompts processed:
        return END
    Else:
        return "agent"
    """
    # TODO: Implement this condition
    pass


# Build graph structure
graph = StateGraph(AgentState)
graph.add_node("agent", agent_node)
graph.set_entry_point("agent")

# TODO: Add conditional edge
# graph.add_conditional_edges("agent", should_continue)

# TODO: Compile graph with max_recursion_limit=100
compiled_graph = None



# ---------------------------------------------------------
# 4. TODO: Implement the runtime
# ---------------------------------------------------------

def main():
    """
    Steps:
    - Load prompts from prompts.txt
    - Build initial state
    - Call graph.invoke ONCE
    - Print all outputs
    """

    # TODO: Read prompts
    # TODO: Create initial_state
    # TODO: final_state = compiled_graph.invoke(initial_state)
    # TODO: Print final_state outputs

    pass


if __name__ == "__main__":
    main()
```

📁 prompts.txt (Example)

```
Write a haiku about mountains.
Explain how a carburetor works.
Summarize the plot of Romeo and Juliet.

Explain finite automata to a 5th standard school student.
```

✨ Expected Output (Example)

```
===== FINAL OUTPUTS =====

PROMPT: Write a haiku about mountains.
OUTPUT: Silent peaks arise / cold winds speak through ancient stone / dawn
paints silver light.

PROMPT: Explain how a carburetor works.
OUTPUT: A carburetor mixes air and fuel…

PROMPT: Summarize the plot of Romeo and Juliet.
OUTPUT: Two star-crossed lovers…
```

🎯 Optional Extensions

To deepen student skills, assign any of the following:

⭐ Extension A: Add Tool Use

Add a calculator tool or search tool as part of the agent.

⭐ Extension B: Streaming

Modify the graph to stream intermediate state updates.

⭐ Extension C: Add Memory

Store long-term memory in an external vector store.

⭐ Extension D: Multi-Agent Upgrade

Transform the single agent into:

- A Planner Agent
- An Executor Agent

⭐ Extension E: Add Logging

Record each step to a JSON file for post-run analysis.
