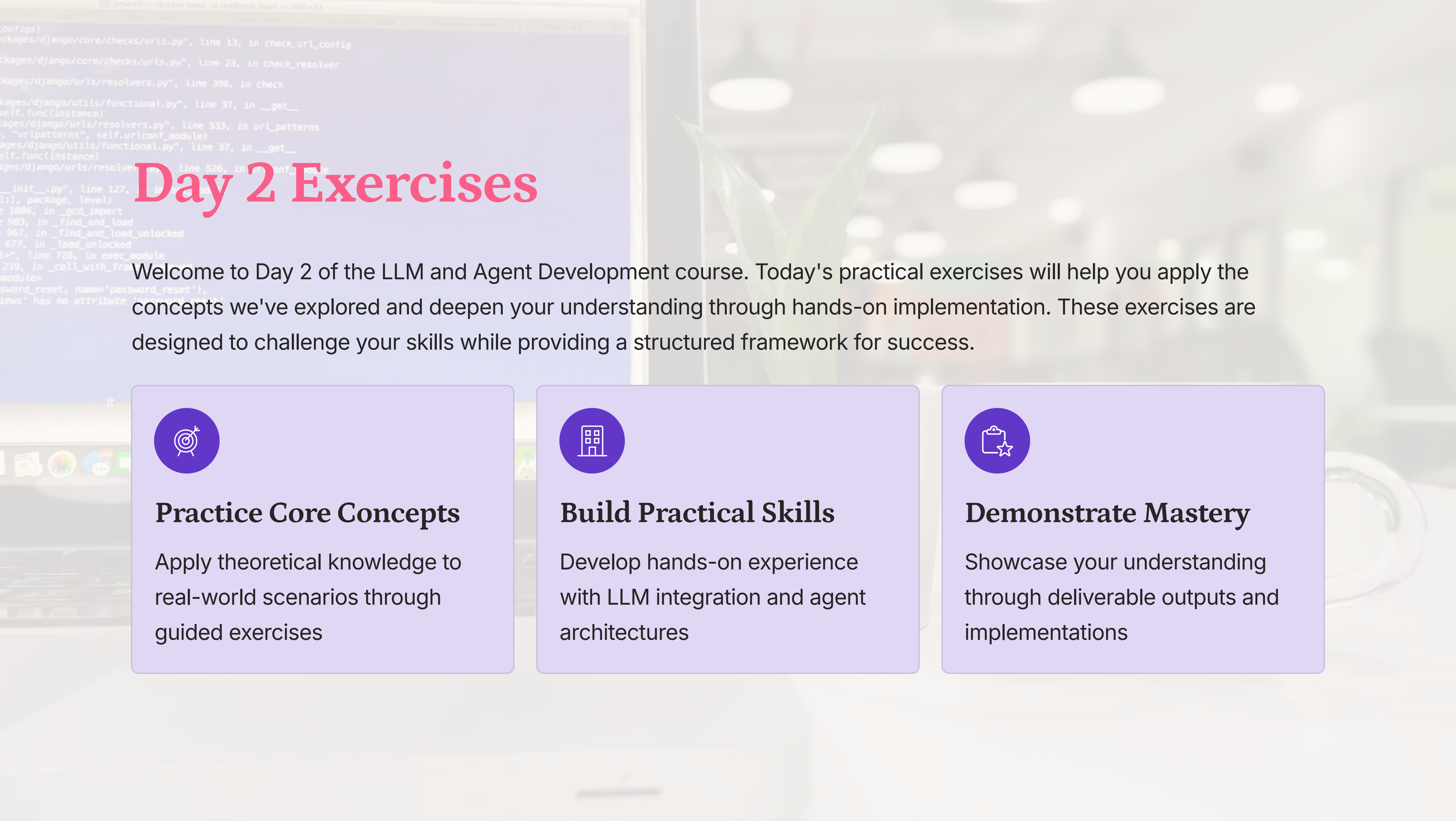

Day 2 Exercises

Welcome to Day 2 of the LLM and Agent Development course. Today's practical exercises will help you apply the concepts we've explored and deepen your understanding through hands-on implementation. These exercises are designed to challenge your skills while providing a structured framework for success.

Practice Core Concepts

Apply theoretical knowledge to real-world scenarios through guided exercises

Build Practical Skills

Develop hands-on experience with LLM integration and agent architectures

Demonstrate Mastery

Showcase your understanding through deliverable outputs and implementations

Exercise 1: MCP Server

MCP Server with one tool: search

Based on demo_code/mcp/basic.py

- Keep only one tool: call the tool **search**
- Tool with two parameters: **query** and **num_results**
- Return some dummy results for now
- Remove run_client_example()
- In main() keep only the mcp.run() code but ensure server uses streamable-http
- Run the server
- Use MCP Protocol Inspector to test the MCP server (run **npx @modelcontextprotocol/inspector**; requires Node.js)

See streamable.py for information required by the run() method

If you want to add authentication via a HTTP header, see search_agent/mcp_server for an example: adds Starlette middleware to the FastMCP server.

Azure AI Search Prerequisites

- Azure AI Search instance accessible by all users (do not use **Free Tier**)
- Azure OpenAI resource with **text-embedding-3-large** embedding model
- Storage Account accessible by all users

Exercise 2: Azure AI Search

Create an index

We will use the **Import data (new)** wizard

- Create a container in the storage account with a unique name
- Upload a number of PDFs or DOCX files to the container
- In the shared Azure AI Search instance, use the **Import data (new)** wizard with **Azure Blob Storage** as the data source and follow the steps
  - You will need access to the Azure OpenAI resource and select the embedding model
- When the wizard asks for an index name, use a **unique name** on the AI Search instance

This should create an indexer, data source, skillset and index. Use **Search Explorer** with your index to check for documents.

Note: the skillset use a split skill and embedding skill to convert documents, split them and create vectors for each split or chunk

Exercise 3: Modify the MCP Server

Modify the search tool to search your index

Use aisearch/query_examples.py for sample search code

- Search your index using hybrid search + semantic reranking
- You can hard code the index name, search keys etc... in the code
- Optional: filter the query based on semantic reranker score
- Test the tool in MCP Inspector

Exercise 4: Create a search agent

Create a search agent that uses the MCP tool

See search_agent folder for inspiration

- You can choose how you create the agent:
  - Responses API:
    - Built-in mcp functionality: requires exposing your MCP server on the Internet (e.g. ngrok)
    - Use a function tool: custom function can use FastMCP MCP client to run the tool on the local MCP server
  - Azure AI Foundry Agent:
    - Built-in mcp functionality: similar to Responses API; requires public endpoint
    - Use a function tool: similar to Responses API
  - LangChain Agent (or other frameworks)
    - MCP functionality is not implemented at the API level
    - This means that your server **does not** require a public endpoint
    - See **search_agent/langchain_agent/search_agent_mcp.py**: uses native support for MCP in LangChain

LangChain recommended