LEARNING PATH

GenAI Development with JavaScript, Python & Azure AI Foundry

Duration: 5 days

COURSE OVERVIEW

This course takes you from the basics of JavaScript and Python to building generative AI solutions with Azure AI Foundry. You will start with the core skills of both languages, such as variables, functions, async code and calling APIs, and then use them to work with large language models deployed in Foundry. Step by step, you will learn prompt engineering, chat with memory, streaming, structured output, tool calling, embeddings and retrieval-augmented generation (RAG) with Azure AI Search, and you will build agents and evaluate their answers. Every part ends with hands-on practice, and the course closes with the best practices for using GenAI on Azure safely, securely and cost-effectively.

LEARNING OUTCOMES

- Write clean JavaScript and Python code, including async code, modules and API calls
- Call models deployed in Azure AI Foundry from both languages, using REST, the `openai` SDK
- Build GenAI features such as chat with memory, streaming, structured output and tool calling
- Create RAG solutions that answer questions from your own documents using embeddings and Azure AI Search
- Build and evaluate AI agents, and apply best practices for safety, monitoring, rate limits and key security

PREREQUISITES

- Basic computer skills: installing software, managing files and folders
- Comfort using a terminal or command prompt to run simple commands
- Basic programming logic, such as what a variable, condition or loop is (no prior JavaScript or Python needed)
- An Azure subscription with access to Azure AI Foundry (free or student credits are enough to start)
- A computer with internet access, and the willingness to learn by building

01	FOUNDATIONS **JavaScript Basics**	10 topics
02	FOUNDATIONS **Python Basics**	10 topics

03	GENERATIVE AI **GenAI with JavaScript + Azure Foundry**	14 topics
04	GENERATIVE AI **GenAI with Python + Azure Foundry**	12 topics
05	SHIPPING **Best Practices on Azure**	4 topics

PART 1: FOUNDATIONS

JavaScript Basics

1.1 Setup: Node.js, VS Code, npm
1.2 Variables, data types and operators
1.3 Conditions and loops
1.4 Functions and arrow functions
1.5 Arrays and objects (map, filter, destructuring)
1.6 Modules (`import` / `export`)
1.7 Async JavaScript: Promises, `async` / `await`
1.8 Calling APIs with `fetch` and working with JSON
1.9 Environment variables with `dotenv`

PART 2: FOUNDATIONS

Python Basics

2.1 Setup: Python, pip, virtual environments (`venv`)
2.2 Variables, data types and operators
2.3 Conditions and loops
2.4 Functions and lambdas
2.5 Lists, dictionaries, tuples and sets
2.6 Modules, packages and `pip install`
2.7 File handling and JSON
2.8 Error handling (`try` / `except`)
2.9 Calling APIs with `requests` and `python-dotenv`

PART 3: GENERATIVE AI

GenAI with JavaScript + Azure Foundry

3.1	LLM basics: tokens, prompts, temperature, context window
3.2	Calling your Foundry model with `fetch` (done: your azure-llm-clean project)
3.3	Prompt engineering: system prompts, examples, output format
3.4	Chat with memory: keeping message history
3.5	Streaming responses word by word
3.6	Structured output: getting JSON from the model
3.7	Tool / function calling
3.8	Embeddings with your Foundry embedding deployment
3.9	RAG with Azure AI Search (`@azure/search-documents`)
3.10	Using different catalog models (GPT, DeepSeek, Llama, Mistral) with the same code
3.11	LangChain.js with Azure (`AzureChatOpenAI`)
3.12	Project: a chatbot web app (Node.js backend + HTML page) using Foundry

PART 4: GENERATIVE AI

GenAI with Python + Azure Foundry

4.1	Calling your Foundry model with `requests` and the `openai` package
4.2	Secure login with Entra ID (`azure-identity`)
4.3	Prompt engineering in practice
4.4	Chat with memory and streaming
4.5	Structured output with Pydantic
4.6	Tool / function calling
4.7	Embeddings and RAG with Azure AI Search (`azure-search-documents`)
4.8	Chat with your PDFs and documents
4.9	Foundry Agent Service: building agents with the `azure-ai-projects` SDK
4.10	Evaluating answers with `azure-ai-evaluation` (quality and safety checks)
4.11	Frameworks: LangChain (`AzureChatOpenAI`) or Semantic Kernel
4.12	Project: a document Q&A app with Streamlit, powered by Foundry

PART 5: Best Practices

Best Practices on Azure

5.1 Content safety and responsible AI filters in Foundry

5.2 Monitoring usage, cost and logs

5.3 Handling rate limits (429 errors) and retries

5.4 Keeping keys safe: `.env`, Azure Key Vault, managed identity