Python AI Development

Building Intelligent, Agentic, and Responsible AI Solutions on Azure

Course Duration: 32 Hours (4 Days) | Audience: Python Developers & AI Engineers

Overview

This four-day hands-on program equips Python developers and AI engineers with the practical skills to design, build, and deploy intelligent AI solutions on Microsoft Azure. The course covers the full spectrum of modern AI development — from Azure OpenAI and LLM fundamentals through multimodal AI, vector search, retrieval-augmented generation, agentic AI, multi-agent orchestration, and responsible AI practices using Microsoft AI Foundry and the Microsoft Agent Framework.

Every day is structured around a focused set of topics with dedicated lab time. Participants work entirely in Python throughout, using pre-configured lab environments. The emphasis is on building real, working solutions; not slides and theory.

Audience Profile

This course is designed for Python developers, AI engineers, and solution architects who want to build production-grade AI applications on Azure. Participants should be comfortable writing Python code and working with REST APIs. Prior Azure experience is helpful but not required — labs use pre-configured environments.

Prerequisites

- Proficiency in Python including functions, classes, and working with JSON and REST APIs
- Basic familiarity with cloud computing concepts
- No personal Azure subscription needed — labs run in pre-configured environments
- VS Code or similar IDE familiarity recommended

What You Will Be Able to Do After This Course

- Connect to and call Azure OpenAI models from Python using the Azure OpenAI SDK
- Apply prompt engineering techniques and use function calling for structured AI outputs
- Process images and documents using Azure multimodal AI capabilities
- Build vector search indexes and implement RAG pipelines grounded in enterprise data
- Build single AI agents with tools and memory using Semantic Kernel in Python
- Orchestrate multi-agent systems with role-based coordination using the Microsoft Agent Framework
- Deploy and evaluate AI solutions using Microsoft AI Foundry
- Apply responsible AI controls including content safety and output verification

Course Syllabus

Day 1: Azure AI Foundry, Azure OpenAI, and LLM Fundamentals

Module 1: The Azure AI Platform and Microsoft AI Foundry

- Overview of the Microsoft AI platform: Azure AI Services, Azure OpenAI, and Azure AI Foundry
- AI Foundry hubs and projects: the unified environment for building and deploying AI solutions
- Deploying a model in AI Foundry: model catalog, deployment types, and connecting from Python
- Authentication: API keys vs. managed identity — how to connect securely from Python code
- Navigating the AI Foundry portal: playground, deployments, connections, and monitoring
- Lab: Create an AI Foundry project, deploy GPT-4o, and make your first API call from Python

Module 2: How Large Language Models Work

- What LLMs are: transformers, pre-training, and why they can generate human-like text
- Tokenization: how text becomes tokens and why it affects prompt design and cost
- Context windows: limits, conversation history management, and trade-offs
- Key parameters: temperature, top-p, max tokens — what they do and when to change them
- Model families on Azure OpenAI: GPT-4o, GPT-4o mini, o-series reasoning models, and embedding models
- Lab: Experiment with parameters in the AI Foundry playground and observe output differences

Module 3: Chat Completions and Prompt Engineering in Python

- The chat completions API: system messages, user messages, assistant turns, and conversation history
- Prompt engineering techniques: zero-shot, few-shot, chain-of-thought, and role prompting
- Streaming completions: handling real-time token-by-token responses in Python
- Managing conversation state across multiple turns in a Python application
- Token counting and cost estimation using the tiktoken library
- Lab: Build a multi-turn streaming chat application in Python with conversation history

Module 4: Function Calling and Structured Output

- What function calling is: extending LLMs with the ability to trigger external tools and APIs
- Defining tools in Python and passing them to the Azure OpenAI API
- Handling tool call responses: parsing arguments, executing the function, and returning results
- Structured output mode: forcing the model to return JSON matching a defined schema
- Real-world pattern: connecting an LLM to a live data source through function calling
- Lab: Build a Python assistant that calls a live REST API via function calling and returns structured results

Day 2: Multimodal AI, Vector Search, and RAG Patterns

Module 5: Multimodal AI — Images and Documents

- Multimodal capabilities in GPT-4o: sending images alongside text in the chat completions API
- Processing image inputs in Python: base64 encoding vs. URL references
- Visual question answering: using GPT-4o to reason over images and answer natural language questions
- Azure AI Document Intelligence: extracting structured fields from invoices, forms, and PDFs

- Combining Document Intelligence and OpenAI: extract then summarize or explain
- Lab: Build a Python pipeline that accepts an image and a PDF, analyses both using GPT-4o and Document Intelligence, and produces a structured summary

Module 6: Vector Embeddings and Semantic Search

- What embeddings are: converting text into high-dimensional vectors that capture semantic meaning
- Generating embeddings in Python using Azure OpenAI embedding models
- Cosine similarity: measuring how semantically close two pieces of text are
- Azure AI Search: setting up an index with vector search fields in Python
- Hybrid search: combining keyword (BM25) and vector similarity for better retrieval results
- Lab: Generate embeddings for a document set, index them into Azure AI Search, and run semantic queries from Python

Module 7: Building a RAG Pipeline

- What RAG is and why it solves hallucination and knowledge cutoff problems in LLMs
- The RAG pipeline end-to-end: chunk, embed, index, retrieve, augment, generate
- Chunking strategies: fixed-size, paragraph-based — trade-offs for retrieval quality
- Constructing the augmented prompt: injecting retrieved context into the system message
- Source attribution: returning document references alongside the generated answer
- Lab: Build a complete RAG pipeline in Python that answers questions grounded in a document library with source citations

Module 8: Improving RAG Quality

- Query rewriting: using an LLM to reformulate the user's question before retrieval
- Re-ranking results: scoring retrieved chunks for relevance before passing them to the model
- Evaluating RAG output: groundedness, relevance, and answer completeness
- Handling no-answer scenarios: when retrieved context does not support a confident answer
- Common RAG failure modes and how to diagnose them
- Lab: Extend the Day 2 RAG pipeline with query rewriting and basic groundedness evaluation

Day 3: Agentic AI and Multi-Agent Orchestration

Module 9: Introduction to Agentic AI

- What agentic AI is: agents, tools, memory, and autonomous goal-directed behaviour
- The agent loop: perceive, reason, select a tool, act, observe the result, iterate
- When to use an agent vs. a direct LLM call: multi-step tasks and external tool needs
- The Microsoft agentic stack: Semantic Kernel for single agents, Microsoft Agent Framework for multi-agent systems
- Responsible agentic AI: setting autonomy limits, human-in-the-loop, and guardrails

Module 10: Building Agents with Semantic Kernel

- Semantic Kernel architecture: kernel, plugins, native functions, and the function calling loop
- Setting up Semantic Kernel in Python and connecting it to Azure OpenAI
- Creating native plugins: wrapping Python functions as tools the agent can call
- Agent memory: conversation history and connecting to a vector store for knowledge retrieval
- Stepwise reasoning (ReAct pattern): the agent iterates through reasoning and tool use until the goal is met
- Lab: Build a Python agent with Semantic Kernel that uses a RAG knowledge plugin to answer questions grounded in a document library

Module 11: Multi-Agent Orchestration with Microsoft Agent Framework

- What the Microsoft Agent Framework is: AutoGen-based multi-agent coordination on Azure
- Core concepts: agents, group chat, message passing, and conversation termination
- Defining agent roles: orchestrator, specialist, and critic agents — each with its own persona and system prompt
- Group chat patterns: how agents take turns, how the orchestrator routes tasks to specialists
- Human-in-the-loop: pausing the agent conversation to request user approval before proceeding
- Lab: Build a three-agent Python system — an orchestrator delegates a task to a specialist, a critic reviews the output, and a human approves before the final response is returned

Module 12: Connecting Agents to Enterprise Data and Services

- Giving agents access to Azure AI Search for RAG-powered knowledge retrieval as a plugin
- Connecting agents to external data via Python tool functions: databases, APIs, SharePoint
- Calling Microsoft Graph API from an agent: reading emails and Teams messages as context
- Logging and observability: tracing the agent conversation and all tool calls for debugging
- Security considerations: managed identity and least-privilege access for agent tools
- Lab: Extend the multi-agent system with a SharePoint data lookup tool and full conversation trace logging

Day 4: Microsoft AI Foundry, Responsible AI, and Capstone

Module 13: Deploying Solutions with Microsoft AI Foundry

- AI Foundry prompt flows: building, testing, and versioning LLM pipelines as deployable artifacts
- Connecting data sources to a Foundry project: Azure AI Search, Azure Storage, and custom APIs
- Deploying a prompt flow as a managed endpoint and calling it from Python
- Monitoring in Foundry: token usage, latency, error rates, and safety filter events
- Managing secrets securely: Azure Key Vault integration for API keys and connection strings
- Lab: Package the Day 3 RAG agent as a Foundry prompt flow, deploy it as an endpoint, and call it from a Python client

Module 14: Responsible AI — Safety, Content Filtering, and Evaluation

- Microsoft responsible AI principles: fairness, reliability, privacy, transparency, and accountability
- Azure AI Content Safety: detecting harmful content categories in inputs and outputs from Python

- Configuring content filters on Azure OpenAI deployments: severity levels and block policies
- Prompt injection detection: identifying adversarial inputs designed to override system instructions
- Groundedness detection: identifying responses not supported by the retrieved context
- Lab: Add content safety filtering and groundedness detection to the capstone agent pipeline

Module 15: Evaluating AI Solutions in AI Foundry

- Why evaluation matters: moving from manual testing to systematic quality measurement
- Built-in AI Foundry evaluators: groundedness, relevance, coherence, fluency, and safety
- Running a batch evaluation in Foundry: submitting a labelled dataset and reviewing results
- Custom evaluators: writing Python evaluation functions for domain-specific quality checks
- Using evaluation results to improve: identifying weak modules and iterating on prompts or retrieval
- Lab: Run a full evaluation of the capstone solution using AI Foundry built-in and custom Python evaluators

Module 16: Capstone Project

- Capstone brief: build a complete end-to-end AI solution combining skills from all four days
- Required components: Azure OpenAI for generation, Azure AI Search for RAG, a Semantic Kernel or Agent Framework agent, and AI Foundry for deployment
- Responsible AI requirement: integrate at least one content safety or groundedness control
- Each participant or pair presents: a live demonstration, a walkthrough of the Python code, and the evaluation results
- Trainer and peer feedback, key takeaways, and recommended next steps for continued learning

Conclusion and Certification

- Participants who complete this program will receive a Certificate of Completion from Koenig Solutions
- This course provides strong foundational coverage aligned with Microsoft AI-103 (Azure AI Engineer) objectives for candidates who wish to pursue certification as a next step
- Recommended Microsoft Learn paths for continued practice: Build generative AI solutions with Azure OpenAI, Develop AI agents on Azure, Implement knowledge mining with Azure AI Search
- Koenig Solutions offers post-training support.