

Python, Coding Copilots and LLMs

Guided AI-assisted programming and bounded agent prototypes

FA-0660 · 3 days

Software Development · Agentic & Generative AI

Refresh Python, review AI-generated code and build a small prepared LLM application with selected retrieval or tool integration and a supervised agent workflow.

Introduction

This three-day workshop is for learners who already understand basic programming concepts but may be new to Python. It develops a working Python foundation through small scripts, then introduces AI coding assistance and selected LLM application patterns.

The course uses a prepared project scaffold and approved accounts to combine code review, a simple retrieval example, tool connectivity and a bounded agent workflow. Participants practise testing, checking retrieved evidence and reviewing tool actions; generated code and model responses are not assumed correct.

The broad framework catalogue is taught through selected labs and architecture demonstrations. The capstone is a small supervised prototype, not an independently secured production service or a guarantee of productivity gains. Multi-agent orchestration, scaling and production deployment are follow-on topics.

Learning outcomes

The workshop teaches participants to:

- Write and inspect small Python programs using control flow, collections, functions, modules, files and exceptions.
- Use an approved coding assistant for completion, explanation, refactoring and test suggestions, then validate its output.
- Call a configured LLM through a prepared Python application.

- Explain and demonstrate a small retrieval-augmented workflow, checking source relevance and answer support.
- Explain MCP client/server tool connectivity and demonstrate an approved version-compatible integration.
- Build a bounded tool-using agent with constrained capabilities, stopping conditions and review points.
- Integrate selected components into a small capstone and report its tests, limitations and remaining operational work.

Prerequisites

- Basic familiarity with programming concepts (variables, functions) though not necessarily in Python.
- A laptop with internet access and the ability to install software / Python packages.
- Interest in AI/ML, coding productivity, and next-gen tooling (no deep ML research background required).
- Willingness to experiment, ask questions, and engage in hands-on tasks rather than pure lecture.

A prepared Python/IDE environment, Git access for the lab, compatible pinned libraries and approved coding-assistant/LLM accounts with any required capacity. Installation rights, credentials, service charges and tool access must be arranged in advance. Use illustrative or authorised data; model-training expertise is not required.

Training outline

1. Day 1 – Python and a Prepared Development Environment

Orientation and setup

- Course overview, objectives, how we'll work.
- Review the prepared Python, IDE, Git and virtual environment; verify the selected dependencies and account access
- Introduction to AI coding assistants: what is "Copilot"-style assistance? Overview of LLMs, code generation, suggestions.

Python foundations

- Basics of Python: syntax, variables, data types (strings, numbers, lists, dicts, sets, tuples).
- Control flow: if statements, loops (for, while), comprehension syntax.
- Functions: defining functions, parameters, return values, default and keyword arguments.

- Modules and packages: importing modules, using standard library modules (e.g., os, sys, json), installing third-party packages with pip.
- Data structures and working with them: list methods, dictionary methods, set operations; iteration patterns.
- File I/O and basic exceptions: reading and writing text and JSON files, handling errors with try/except.
- Basic object-oriented programming: classes, attributes and methods; simple inheritance.
- Working on short mini-projects: for example, a small command-line tool, or simple data processing script.

2. Day 2 – Coding Assistance and LLM Integration

Using a coding assistant critically

- Introduction to AI-assisted coding: what is a “copilot”, how it works, benefits and caveats.
- Use the approved coding-assistant integration and inspect its relevant IDE/account settings
- Completion, function suggestions, refactoring, docstrings and pseudocode conversion; review before accepting
- Specify requirements and constraints, review correctness and security, test generated code and avoid unsupported assumptions
- Hands-on exercises: pair coding tasks where participants generate code with assistance, then review and refine it.
- Debugging and testing: using the assistant to help write unit tests, detect errors, and refactor existing code for clarity.

Selected retrieval and tool-connectivity labs

- RAG: retrieve relevant material to support answers; retrieval does not guarantee factual correctness
- LangChain models, prompts, tools and agents; use APIs compatible with the pinned lab release
- Prepared retrieval exercise: load authorised sample documents, create/query embeddings and a small store, then review the resulting answer against sources
- MCP: a protocol for tools and context, not a guarantee of scalability, permission or trust
- LangChain MCP integration matched to the lab version: current native MCPAdapter or earlier langchain-mcp-adapters where appropriate; newer native namespace is beta
- Connect to an approved small MCP server or inspect a supplied server example; keep tools and data access constrained
- Toolchains: data access, context, prompts, authentication and explicit permission boundaries
- Discussion of architecture: chaining LLM calls, managing context, pipelines vs agents.

3. Day 3 — Bounded Agents and Capstone

Agent concepts and a constrained workflow

- Definition: what is an “agent” in LLM/dev-context? A system that uses an LLM to decide actions, pick tools, and orchestrate workflows.
- Router, reflection, tool-calling and multi-agent patterns as comparisons; implementation focuses on one bounded agent
- Build a constrained LangChain tool-calling example using an approved lab tool
- Choose between approved read-only lab tools, collect results and respond with reviewable evidence
- Context limits, bounded iterations, monitoring and stopping conditions; never assume autonomous reasoning is reliable
- Multi-step orchestration, memory and asynchronous invocation as demonstrations or extension ideas

A small integrated project

- Use a small scaffolded project combining Python, reviewed coding assistance, an LLM and selected RAG/MCP/tool components
- Project planning: define scope, choose tool(s) or domain (e.g., document Q&A, code documentation assistant, data summariser, interactive chatbot).
- Implement a bounded core workflow using selected prepared components, not every architecture option
- Review & refine: peer-review sessions, instructor feedback, testing.
- Short project demonstration, observed test results, limitations and follow-on work

Review and follow-on work

- Recap of key takeaways.
- Review next learning priorities rather than a mandatory reference or resource list
- Follow-on considerations: deployment, scaling, security/privacy and advanced architectures, not completed production deliverables
- Q&A and feedback.