Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Building AI Agents with Python and MCP

Class Duration

35 hours of live training delivered over 5 days.

Student Prerequisites

- Professional Python development experience
- Familiarity with LLM APIs: prompts, tool/function calling, and structured outputs (see *LLM Application Development with Python*)
- Comfort with async programming (`asyncio`) and Git
- No prior agent framework experience required

Target Audience

Python developers ready to move beyond single request/response LLM calls to autonomous, multi-step systems. Ideal for engineers building internal agents that plan work, call tools, and act on company data and APIs - and for platform teams standardizing how agents connect to internal systems via the Model Context Protocol. Includes a dedicated module on running open-weight models locally for organizations with data residency, privacy, or AI usage restrictions.

Description

Agents are the defining LLM application pattern of 2026: systems that take a goal, plan steps, call tools, observe results, and iterate until done. This five-day intensive teaches Python developers to build them for production, starting with a from-scratch agent loop so every later abstraction is understood rather than trusted. From there the course covers the leading frameworks, LangGraph 1.x and Pydantic AI v1, then the Model Context Protocol: building, securing, and testing MCP servers in Python with the official SDK and FastMCP, and consuming them from your own agents and from coding assistants like Claude Code and Cursor. The final days cover multi-agent orchestration, human-in-the-loop design, and the evaluation discipline agents demand, then production realities: security, observability, cost control, deployment patterns, and running open-weight models locally with Ollama and vLLM for restricted environments. The course assumes working LLM API skills (covered in *LLM Application Development with*

Python); developers who want a broad generative AI grounding first should start with *Generative AI and LLMs for Python Programmers*.

Learning Outcomes

- Explain the agent loop (plan, act, observe, iterate) and implement it from scratch in Python with tool dispatch, working memory, and stop conditions.
- Design tool interfaces, schemas, and error contracts that models invoke reliably, and use structured outputs for plans, routing decisions, and validated results.
- Build stateful agent workflows in LangGraph 1.x and type-safe agents with Pydantic AI v1, and choose the right framework (or plain Python) for a given problem.
- Build, secure, and test MCP servers in Python with the official SDK and FastMCP that expose internal tools, data, and APIs.
- Connect MCP servers to your own agents and to coding assistants like Claude Code and Cursor.
- Design multi-agent systems and human-in-the-loop workflows: planner/worker patterns, hand-offs, approval gates, and failure recovery.
- Evaluate agents with trajectory evals, tool-call accuracy metrics, golden tasks, and regression suites in CI, and apply prompt-injection defenses, tool sandboxing, and least-privilege credentials.
- Deploy agents as observable, cost-controlled services, and run open-weight models locally with Ollama and vLLM for restricted environments.

Training Materials

Comprehensive courseware is distributed online at the start of class. All students receive a downloadable MP4 recording of the training.

Software Requirements

Python 3.13+, an Anthropic API key (instructions for OpenAI and Gemini also provided), Ollama installed locally, VS Code or an editor of choice, Docker or Podman, and Git.

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Training Topics

Environment Setup and SDK Configuration

- Python 3.13+ project setup for agent development
- Anthropic, OpenAI, and Gemini Python SDKs: installation and authentication
- Sync vs. async clients for agent workloads
- API keys, environment configuration, and secret hygiene
- Choosing models across the Claude 5/4.x, GPT-5.x, and Gemini 3.x families
- Why agents multiply token costs - and what that means for design

The Agent Loop from Scratch

- The agent loop: plan, act, observe, iterate
- Anatomy of a single turn: messages, tool definitions, and stop reasons
- The message list as agent state
- Stop conditions, budgets, and runaway prevention
- Iteration limits, timeouts, and cost ceilings
- What frameworks abstract - and what they cost you

Tool Dispatch and Execution

- Tool dispatch and structured tool results
- Mapping tool calls to Python functions: registries and decorators
- Handling parallel tool calls and result ordering
- Serializing results back to the model: text, JSON, and images
- Exceptions vs. error results: what the model should see
- Concurrent tool execution with `asyncio`

Tool Design for Agents

- Tool schemas models use reliably
- Names and descriptions as prompt engineering
- Granularity: few powerful tools vs. many narrow ones
- Error contracts and recoverable failures
- Idempotency and side-effect safety
- Pagination, truncation, and response-size limits

Structured Outputs Inside Agents

- Pydantic models as agent result contracts

- Structured plans, routing decisions, and classifications in the loop
- JSON Schema-constrained generation vs. free-form reasoning
- Validation failures and re-prompting strategies
- When to constrain output - and when to let the model write prose

Context and Memory Management

- Working memory and conversation state
- Context window budgets and compaction strategies
- Summarizing and pruning tool results mid-run
- Long-term memory: persistent stores across sessions
- Episodic memory: retrieving relevant past interactions
- Scratchpads, notes, and file-backed memory patterns

Agent Frameworks: LangGraph Core

- LangGraph 1.x: graph-structured workflows with nodes, edges, and typed state
- Graphs vs. free-running loops: when structure pays off
- State channels, reducers, and partial updates
- Checkpointing and resumable runs
- Persistence backends and thread management
- Replaying runs for debugging

LangGraph: Streaming, Interrupts, and Subgraphs

- Streaming agent progress to clients
- Stream modes: tokens, state updates, and events
- Human-in-the-loop interrupts and approvals
- Resuming interrupted runs with reviewer input
- Subgraphs: composing larger systems from smaller graphs
- Durable execution for long-running agents

Agent Frameworks: Pydantic AI

- Pydantic AI v1: type-safe agents with dependency injection
- Agents, tools, and typed run contexts
- Structured outputs as agent results
- Model-agnostic agents across providers
- Testing agents with overridden dependencies
- Fast unit tests with stub and function models

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Choosing an Agent Framework

- Framework selection: LangGraph vs. Pydantic AI vs. plain Python
- Decision criteria: control, state needs, team skills, and ecosystem
- Provider agent SDKs and where they fit
- Avoiding lock-in: keeping tools and prompts portable
- Reading framework source: knowing what runs under you

Building MCP Servers in Python

- MCP architecture: hosts, clients, and servers
- Building servers with the official Python SDK and FastMCP
- Defining tools with typed parameters and clear descriptions
- Resources for read-only data and prompts for reusable templates
- Transports: stdio for local servers, streamable HTTP for remote
- Structured tool output and error reporting

Securing and Testing MCP Servers

- Authentication and credential scoping
- Authorization for streamable HTTP servers
- Testing and debugging MCP servers with the Inspector
- Unit testing tool logic as plain Python functions
- Versioning and maintaining tool contracts
- Logging and observability for server behavior

Consuming MCP from Assistants and Agents

- Connecting MCP servers to your own agents
- MCP client sessions: discovery, invocation, and lifecycle
- MCP in coding assistants: Claude Code and Cursor
- Combining multiple servers in one agent
- Tool-name collisions and selective tool exposure
- When to wrap an API in MCP - and when to call it directly

Multi-Agent Orchestration

- Planner/worker and supervisor architectures
- Agent hand-offs and shared context
- Specialist agents with scoped tools and prompts
- Parallel agents and result aggregation
- Failure recovery and partial-result handling
- When one agent with good tools beats a multi-agent system

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Human-in-the-Loop Design

- Where humans belong in agent workflows
- Approval gates for destructive actions
- Pausing, resuming, and editing agent state mid-run
- Escalation paths and confidence thresholds
- Review queues and structured approval UX

Evaluating Agents

- Trajectory evals: judging the path, not just the answer
- Tool-call accuracy and efficiency metrics
- Golden tasks and regression suites in CI
- Building eval datasets from production transcripts
- LLM-as-judge for agent transcripts
- Handling nondeterminism: repeated runs and flaky tasks

Security and Guardrails

- Prompt injection via tool results and retrieved content
- The dangerous combination: private data, untrusted content, and egress
- Tool sandboxing and least-privilege credentials
- Input/output filtering and policy checks
- Approval gates for destructive actions
- Audit trails for agent actions

Local and Restricted Environments

- Open-weight models: the current landscape
- Serving locally with Ollama and vLLM
- Hardware sizing, quantization, and throughput trade-offs
- Tool calling with local models: capabilities and limits
- OpenAI-compatible endpoints: swapping backends without rewrites
- Architecting agents for data-residency and compliance constraints

Observability and Cost Control

- Tracing agent runs: spans for turns, tool calls, and model calls
- Token budgets and cost attribution
- Prompt caching for repetitive agent context
- Dashboards and alerting on cost, latency, and failure rates
- Capturing transcripts for debugging and future evals

To discuss this course and customizations:
Call: +1 434-509-6890 or Email: sales@cloudcontraptions.com

Deployment Patterns

- Deploying agents as services
- Containerizing agent workloads
- Queues and workers for long-running tasks
- Status reporting: polling, webhooks, and progress streams
- Graceful failure handling and runaway prevention in production
- Scaling: concurrency, provider rate limits, and quotas

