Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

AI-Assisted Development with Python

Class Duration

35 hours of live training delivered over 5 days.

Student Prerequisites

- Professional Python development experience
- Working knowledge of Git and GitHub (issues, pull requests, branches, reviews)
- Comfort with the command line and a virtual environment workflow
- No prior AI tool experience required

Target Audience

Professional Python developers who want a structured, end-to-end path from AI-curious to AI-proficient. Ideal for engineers and teams adopting AI coding tools across the full development lifecycle — writing, refactoring, testing, reviewing, and shipping Python code — who also need to understand the security, licensing, and governance implications of doing so responsibly. This is a tool-agnostic course: students work hands-on with Claude Code, GitHub Copilot, Codex, Cursor, Windsurf, and Gemini CLI rather than betting on a single vendor, and the skills transfer to whichever assistant your organization adopts.

Description

AI coding tools have moved from autocomplete novelty to a core part of the professional Python workflow — but most developers use a fraction of their capability and few teams have habits for using them safely. This five-day intensive builds those habits from the ground up. Days one and two cover the foundations: how large language models generate code, the strengths and tradeoffs of the frontier models, and prompting and context engineering techniques that reliably produce idiomatic, type-annotated, modern Python. Days two and three move into the tools: inline assistance and chat-driven editing in the IDE, then full agentic workflows across Claude Code, GitHub Copilot, Codex, Cursor, Windsurf, and Gemini CLI — plan-then-execute sessions, grounding agents with project context files, and issue-to-PR automation. Day four applies AI to quality: generating and maintaining

pytest suites, AI-assisted debugging, code review, and refactoring legacy Python. Day five addresses what teams need to adopt AI at scale: security pitfalls (hallucinated packages, prompt injection, secret leakage), licensing and IP questions, team policy design, and honest productivity measurement. Students finish with a capstone that exercises the full workflow on a realistic Python codebase.

Learning Outcomes

- Explain how LLMs generate code and choose the right frontier model for a given Python task.
- Apply prompting and context engineering techniques that produce idiomatic, type-annotated Python.
- Use IDE assistants (Copilot, Cursor, Windsurf) effectively for completion, chat, and multi-file edits.
- Drive agentic coding sessions with Claude Code, Copilot, Codex, Cursor, Windsurf, and Gemini CLI: plan, execute, review, iterate.
- Ground agents in project context with CLAUDE.md / AGENTS.md, rules files, and MCP servers.
- Generate, evaluate, and maintain pytest test suites with AI assistance.
- Use AI for debugging, code review, and incremental refactoring of legacy Python code.
- Recognize Python-specific AI failure modes: hallucinated packages, outdated idioms, and subtle type errors.
- Apply security guardrails: dependency vetting, secret handling, and prompt-injection awareness.
- Contribute to team standards for licensing, governance, and measuring AI's real productivity impact.

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+ (3.14 recommended), a GitHub account with Copilot access, Claude Code CLI with an Anthropic API key or subscription, Codex, Cursor, Windsurf, and the Gemini CLI (trials and free tiers acceptable), VS Code, Git, and the GitHub CLI (`gh`).

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Training Topics

Foundations: How AI Writes Code

- How LLMs generate code: tokens, context windows, and training data
- The frontier models for coding: Claude, GPT, and Gemini compared
- Capability tradeoffs: speed, cost, context size, and reasoning
- Where AI excels in Python work and where it reliably fails

Prompting and Context Engineering for Python

- Writing prompts that produce idiomatic, modern Python (3.12–3.14)
- Steering toward type hints, dataclasses, and current standard-library idioms
- Context strategies: what to include from a large Python codebase
- Structured outputs and constraining generation
- Avoiding outdated idioms and deprecated APIs in generated code

IDE Assistants in Daily Python Work

- GitHub Copilot: completions, chat, edits, and Next Edit Suggestions
- Cursor: Composer, rules files, and project-level context
- Windsurf: Cascade, planning, and memories
- The wider landscape: Aider, Cline, JetBrains Junie, and Amazon Kiro
- Multi-file edits and chat-driven refactoring
- Docstring, type-stub, and documentation generation
- Customizing assistants with instructions files

Agentic Coding: Claude Code, Copilot, Codex, Cursor, Windsurf, and Gemini CLI

- The agentic loop: plan, act, observe, iterate
- The agent landscape compared: Claude Code, Copilot agent mode, Codex, Cursor CLI, Windsurf Cascade, and Gemini CLI
- Plan mode and the supervision spectrum: permission prompts to auto mode
- CLAUDE.md and AGENTS.md: conventions, virtual-env and tooling context
- Running tests and iterating until green: pytest in the agent loop
- Issue-to-PR workflows: assigning work to cloud agents
- MCP servers: connecting agents to internal tools and data

Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

AI-Driven Testing and Quality

- Generating pytest suites: fixtures, parametrization, and coverage gaps
- Property-based testing with Hypothesis and AI assistance
- AI-assisted debugging: tracebacks, logging, and bisection
- Agentic code review and pull request automation
- Refactoring legacy Python: characterization tests first, then change

Python-Specific Pitfalls and Verification

- Hallucinated packages and typosquatting risk in `pip` / `uv` installs
- Subtle type errors: verifying with mypy and pyright
- Linting and formatting generated code with ruff
- Performance review of generated code: profiling before trusting

Security, Licensing, and Governance

- Prompt injection and untrusted content in agent workflows
- Secret leakage: what never goes in a prompt or context file
- Dependency vetting and supply-chain hygiene
- Licensing and IP status of AI-generated code
- Team policy design: review requirements and audit trails
- Measuring real productivity impact honestly

Capstone Workshop

- Plan and execute a multi-step feature on a realistic Python codebase with the coding agent of your choice
- Generate and harden a pytest suite for an untested module
- Issue-to-PR exercise: agent-ready issue through reviewed merge
- Security review of an AI-generated changeset
- Q&A session