Cloud Contraptions
TRAINING & CONSULTING

To discuss this course and customizations:
Call: +1 434-509-6890 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: each delivery is tailored to the AI tools your organization chooses - Claude Code, GitHub Copilot, Codex, Cursor, Windsurf, Gemini CLI, or others - and the skills transfer to whichever assistants your team 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, starting with how large language models generate code, setting up and configuring whichever tools your organization chooses, and prompting and context engineering techniques that reliably produce idiomatic, type-annotated, modern Python. Daily practice follows: disciplined inline assistance, chat-driven explain-refactor-generate workflows, grounding assistants in project context with CLAUDE.md / AGENTS.md and rules files, and full agentic workflows from plan-then-execute sessions to issue-to-PR automation. Day four applies AI to quality across the lifecycle: pytest and Hypothesis suites, AI-assisted

debugging, refactoring at scale behind characterization-test safety nets, legacy modernization, and documentation. Day five addresses what teams need to adopt AI at scale: security pitfalls, licensing and IP questions, organizational policy, cost management, and rolling AI out with honest productivity measurement. This is a tool-agnostic course: each delivery is tailored to the AI tools your organization chooses (Claude Code, GitHub Copilot, Codex, Cursor, Windsurf, Gemini CLI, or others).

Learning Outcomes

- Explain how LLMs generate code, choose the right frontier model for a given Python task, and set up and configure your chosen AI tools with sound privacy and data controls.
- Apply prompting and context engineering techniques that produce idiomatic, type-annotated, modern Python.
- Use inline completions and chat-driven explain, refactor, and generate workflows with discipline: well-scoped context, clean conversation hygiene, and review reflexes.
- Ground assistants and agents in project context with CLAUDE.md / AGENTS.md, rules files, MCP servers, and shared team prompt libraries, and use AI to read and understand unfamiliar codebases.
- Drive agentic coding sessions with your chosen coding agents (plan, execute, review, iterate), including issue-to-PR automation.
- Generate, evaluate, and maintain pytest and Hypothesis suites with AI assistance, and turn tracebacks, async failures, and production logs into root causes faster.
- Refactor at scale and modernize legacy Python behind characterization tests, and generate documentation: docstrings, READMEs, and architecture decision records.
- Recognize Python-specific AI failure modes (hallucinated packages, outdated idioms, subtle type errors), apply security guardrails, and contribute to team standards for cost, licensing, governance, rollout, and measuring 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.

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

Software Requirements

Python 3.13+ (3.14 recommended), access to your organization's chosen AI coding tools (such as GitHub Copilot, Claude Code, Codex, Cursor, Windsurf, or Gemini CLI; trials and free tiers are acceptable), VS Code, Git, and the GitHub CLI.

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
- Setting expectations: assistant, pair programmer, or junior engineer?

Setting Up and Configuring Your Chosen Tools

- Accounts, seats, and enterprise policy: who gets access to what
- Editor and CLI integration for the tools your organization selects
- Model selection and switching: defaults, tiers, and per-task choices
- Privacy and data controls: retention, training opt-outs, and telemetry
- Python-aware configuration: interpreters, virtual environments, and uv
- Verifying your setup: a smoke-test workflow before daily use

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

Everyday Inline Assistance

- Completions as a dialogue: writing code that invites good suggestions
- When to accept, when to reject: building review reflexes
- Comment-driven and docstring-driven prompting for completions
- Type hints as steering: how annotations sharpen suggestions
- Avoiding autopilot: staying in command of code you sign

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

Chat-Driven Development

- Explain, refactor, and generate: the core chat workflows
- Scoping context: selections, files, and symbols - only what the question needs
- Conversation hygiene: when to iterate and when to start fresh
- From chat to edit: applying, reviewing, and adjusting suggested changes
- Interrogating generated Python: edge cases, error paths, and assumptions

IDE Assistants in Daily Python Work

- Deep-dive on your chosen IDE assistants (GitHub Copilot, Cursor, Windsurf, or others): completions, chat, edits, and rules/instructions files
- 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

Working with Project Context

- CLAUDE.md and AGENTS.md in depth: what belongs, what doesn’t
- Rules and instructions files across tools: one convention, many formats
- Encoding repo conventions: style, architecture, and testing norms
- Team prompt libraries: capturing prompts that work and sharing them
- Python project context: pyproject.toml, tooling, and environment notes
- Keeping context files current as the codebase evolves

AI-Assisted Code Reading and Comprehension

- Onboarding to unfamiliar Python codebases with AI guides
- Generating explanations, call graphs, and architecture diagrams
- Tracing data flow and side effects through dynamic Python
- Summarizing modules, packages, and pull requests
- Code archaeology: reconstructing intent from legacy code and history

Agentic Coding with Your Chosen Tools

- The agentic loop: plan, act, observe, iterate
- The agent landscape compared - Claude Code, Copilot agent mode, Codex, Cursor CLI, Windsurf Cascade, Gemini CLI - and how to choose

- 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

AI-Driven Testing in Depth

- Generating pytest suites: fixtures, parametrization, and coverage gaps
- Property-based testing with Hypothesis and AI assistance
- Mocking, async tests, and integration-test generation
- Designing for testability: letting AI surface seams and dependencies
- Mutation-style review: testing the tests AI writes
- Agentic code review and pull request automation
- Maintaining suites over time: updating tests as code changes

AI-Assisted Debugging in Depth

- Turning tracebacks into hypotheses: structured debugging prompts
- Async failures, race conditions, and intermittent bugs with AI help
- Log analysis and instrumenting code for better AI diagnosis
- Bisection and root-cause workflows with agents
- Knowing when the AI is guessing: validating proposed fixes

Refactoring at Scale and Legacy Modernization

- Multi-file refactors: planning, executing, and reviewing with agents
- Characterization tests as the safety net before any change
- Python-specific refactors: type-hint adoption, dataclass conversion, package restructuring
- Upgrading dependencies and Python versions with AI assistance
- Incremental modernization: strangler patterns over big rewrites
- Keeping diffs reviewable: small steps over heroic changes

Documentation and Knowledge Work

- Docstring generation that matches your project’s conventions
- READMEs, how-to guides, and onboarding docs from real code
- Architecture decision records: drafting and maintaining ADRs with AI
- API documentation and Sphinx / MkDocs workflows

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

- Keeping documentation honest: reviewing generated prose for drift

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
- Trust but verify: a review checklist for AI-generated Python

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
- Organizational policy: approved tools, data boundaries, and audit trails
- Review requirements: keeping humans accountable for shipped code

Measuring Impact, Cost, and Team Adoption

- Tokens, model tiers, and budgets: managing cost without blunting value
- Choosing the right model for the task: fast and cheap versus deep and slow
- Re-evaluating the tool landscape: selection criteria, trials, and switching costs
- Rollout patterns that work: champions, guilds, and paired onboarding
- Metrics that matter - and vanity metrics that mislead
- Adoption anti-patterns: mandates, leaderboards, and unreviewed merges

