CAREER PATH 20

Python Developer with AI Tools

Use AI assistants responsibly while building reliable Python software

01 **Prompt** › 02 **Review** › 03 **Test**

What You Will Be Able to Explain

Outcome-focused, not tool-name memorization

01 • Role Lens

Explain how AI tools support Python development without replacing engineering judgment.

02 • Workflows

Build a workflow for code generation, debugging, testing, and refactoring.

03 • Evidence

Create portfolio evidence showing review, tests, and safe use of AI.

AI Can Write Code; You Still Own the System

The workplace pressure behind the career path

Current friction

- Generated code is accepted without review
- Tests cover only happy paths
- Dependencies are copied blindly
- Secrets and edge cases are missed

Why this role matters

- Developers define requirements clearly
- AI suggestions are reviewed and tested
- Type hints and small functions improve quality
- Security and maintainability remain human responsibilities

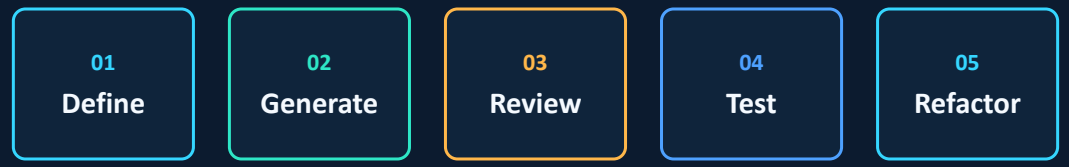
Scenario: InvoiceOps Automation

A small operations team needs a Python tool to process invoice files

The Project Scope

- The tool reads CSV files, validates invoices, flags missing fields and creates a summary report
- AI can draft parsing code, tests and documentation
- The developer must verify data rules, errors, edge cases and security
- The final project includes tests, type hints, README and review notes

Automation Workflow



Key Takeaway

AI speeds up coding only when the developer controls context, quality, and risk.

Role Reality Check

What employers actually expect you to do

Not enough

- Vibe coding only
- Big vague prompts
- No accountability

Workplace evidence

- Requirements + tests + review
- Small well-scoped tasks
- Human-owned code quality

Employer signal

- Python developer expectations
- Better AI output
- Professional standard

Core Foundations

Skills that transfer across tools and platforms

Language Essentials

- Python basics
- Functions
- OOP
- Type hints

Modern Workflow

- File I/O
- APIs
- Git
- pytest

Professional Standards

- Virtual environments
- Debugging
- PEP 8
- Security basics

Good Python with AI starts with strong fundamentals, not prompt shortcuts.

Responsible AI-Assisted Coding Loop

A realistic sequence you can explain in an interview

Step 01

Specify

Step 02

Generate

Step 03

Inspect

Step 04

Test

Step 05

Improve

The loop ends only when code is understandable, tested, secure, and maintainable.

Tools Connected to Tasks

Labels only matter when the task is clear

Task	Tools / Resources	Evidence to Produce
Code drafting	ChatGPT, Copilot, Cursor	*Reviewed function*
Testing	pytest, unittest, mocks	*Test report*
Quality	ruff, mypy, formatting	*Clean checks*
Security	pip-audit, Bandit, review	*Risk notes*

Common Mistakes to Avoid

Small misunderstandings create weak portfolios

Weak Approach

- ✕ Asking AI to build the whole app at once
- ✕ Skipping type hints and input validation
- ✕ Trusting generated explanations
- ✕ Ignoring licenses and dependency risks

Correct Approach

- ✓ Break work into small functions
- ✓ State inputs, outputs, and errors
- ✓ Read and run the code yourself
- ✓ Review packages and keep humans accountable

Practical Lab Path

Build one connected case, not random screenshots

STEP 01
Define spec

STEP 02
Draft function

STEP 03
Write tests

STEP 04
Run checks

STEP 05
Document

The Lab Project: CSV Invoice Validator

Build a robust CSV invoice validator utilizing AI-assisted code. Your final portfolio output must incorporate manual review notes, complete pytest coverage, comprehensive error handling, and a professional README.

Intermediate Capabilities

Where learners become useful to real teams

CAPABILITY 01 **Prompting**	WORKPLACE USE Clear task constraints	VALIDATION **Better draft**
CAPABILITY 02 **Testing**	WORKPLACE USE Validate behavior	VALIDATION **pytest evidence**
CAPABILITY 03 **Refactoring**	WORKPLACE USE Improve maintainability	VALIDATION **Before/after diff**
CAPABILITY 04 **Debugging**	WORKPLACE USE Explain root cause	VALIDATION **Issue note**

Advanced or Emerging Capability

Learn this after the foundation is reliable

Key Focus Areas

- AI-assisted code review with human accountability
- Agentic coding workflows under approval rules
- Secure handling of credentials and files
- Dependency and license review
- Performance profiling and memory-aware refactoring

Python	pytest	unittest
mypy	ruff	Bandit
pip-audit	GitHub	Copilot
Cursor	ChatGPT	uv / pip

Portfolio Evidence

Show decisions, tests, and trade-offs

ARTIFACT	WHAT IT PROVES	INTERVIEW ANGLE
Project spec	Requirement clarity	What should the program do?
Test suite	Quality discipline	What cases prove it works?
Review log	AI accountability	What did you reject or fix?
README	Communication skill	How can someone run it?

U.S. Job-Market Reality Check

Use job-board numbers as dated signals, not guarantees

Reality Check Guidelines

- Counts change by search phrase, duplicate postings, recruiters, and remote filters.
- Search Python Developer, Automation Developer, QA Automation, Data Tools Developer and Backend Python roles.
- AI-tool fluency is useful only when paired with testing, debugging, Git and code review.
- Do not claim AI-generated speed gains unless measured in your own work.

Python Developer

Automation Developer

QA Automation Engineer

Backend Python Developer

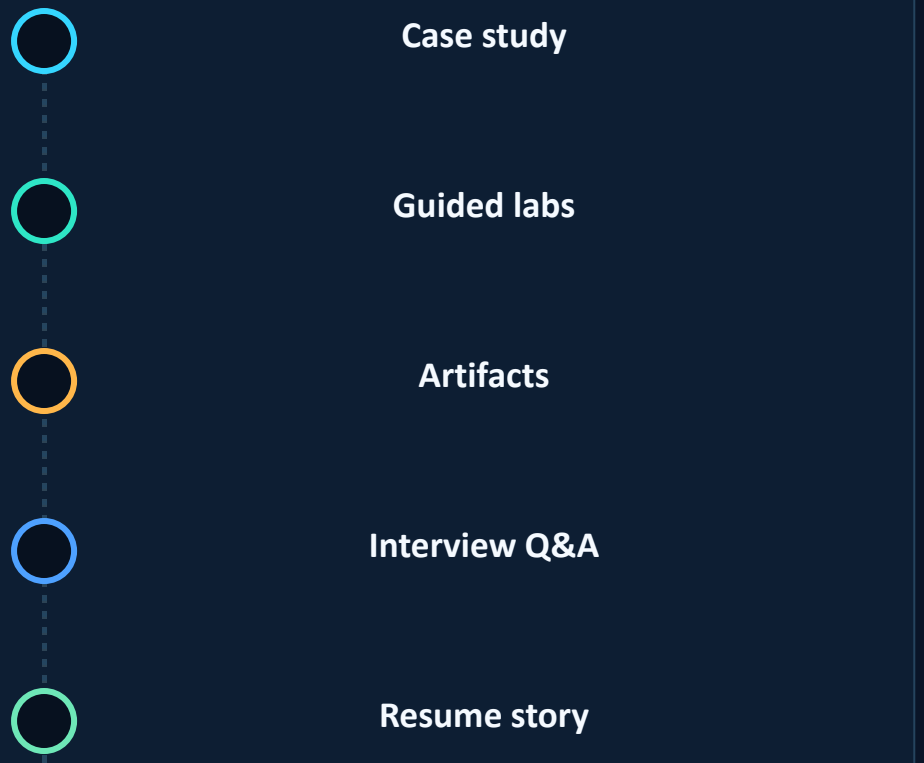
Data Tools Developer

AI-assisted Developer

No salary figures included. Update exact posting counts again on the publish date.

How ITLearn360 Helps You Build Evidence

Workbook + labs + interview practice + portfolio artifacts



CASE STUDY WORKBOOK

InvoiceOps Python + AI workbook

PRACTICAL LABS & EXERCISES

Prompt templates, review checklists, pytest labs and debugging exercises

PORTFOLIO DELIVERABLES

Portfolio-ready project with README, tests and review notes

INTERVIEW PREPARATION

Interview Q&A on AI-assisted coding responsibility

Interview Conversation Practice

Translate your project into clear workplace reasoning

Interviewer asks

“How do you use AI when coding?”

“How do you know generated code is correct?”

“What would you never paste into an AI tool?”

Strong answer includes

- Define the task, generate small pieces, review and test
- Mention unit tests, edge cases, type checks and manual inspection
- Never paste secrets, confidential code or restricted customer data

30-Day Action Plan

A practical start without invented timelines

WEEK 1	**BUILD** Build Python fundamentals	**PROOF** Functions + file I/O
WEEK 2	**BUILD** Use AI for small helpers	**PROOF** Review log
WEEK 3	**BUILD** Add pytest and quality checks	**PROOF** Test report
WEEK 4	**BUILD** Publish portfolio project	**PROOF** README + interview story

ITLearn360 Career Path

Use AI as a coding assistant, not a substitute

Core Developer Guidelines

- Own the requirements
- Review every generated line
- Test normal and edge cases
- Document your decisions