Evaluating the Quality of Python Programming: Course Overview

```
# Better Code
# Cleaner Thinking
# Lasting Impact
```

Why quality matters:
maintainability, defect cost, review throughput, onboarding speed

Six practical dimensions:
correctness, readability, maintainability, testability, performance, security

Python-specific concerns:
dynamic typing, idiomatic style, packaging, dependency hygiene

Roadmap:
dimensions → standards and tooling → review practice → measurement → tech debt → scaling → capstone audit

Audience tracks:
developers, reviewers, tech leads, teachers, advanced learners



HIGHER QUALITY PYTHON | STRONGER SOFTWARE | BRIGHTER POSSIBILITIES

What 'Quality' Means: A Reference Model

BETTER CODE
HIGHER CONFIDENCE
REAL IMPACT

- ISO/IEC 25010:2023 defines nine product quality characteristics
- Key ones: functional suitability, reliability, security, maintainability, flexibility, safety
- Map them to code: readability, analyzability, modifiability, testability
- Practitioner view: six dimensions — readability, maintainability, reliability, security, testability, performance
- Quality is trade-offs, not one number; dynamic typing needs multiple signals



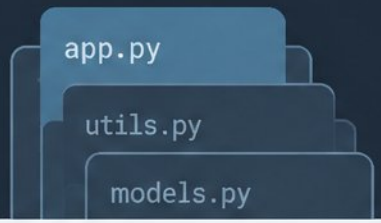
Practitioner view: six dimensions

Readability	Maintainability	Reliability	Security	Testability	Performance
Sign of trouble: Hard to understand	Sign of trouble: Hard to change	Sign of trouble: Fails unexpectedly	Sign of trouble: Vulnerable to attack	Sign of trouble: Hard to test	Sign of trouble: Too slow

Quality is trade-offs, not one number; dynamic typing needs multiple signals

How Quality Defects Show Up in Python Code

COMMON TRAPS. CLEARER CODE. FEWER SURPRISES.



Classic traps:
mutable defaults shared across calls, closure late binding, is vs ==, [[...]] * n aliasing

Bare except:
swallows KeyboardInterrupt and SystemExit; raise ... from err preserves the cause

Module-level mutable state
creates hidden coupling and order-dependent tests

Boundary failures:
empty inputs, None in non-optional code, float equality, assumed non-empty collections

Detection:
Ruff E722 (bare except), Ruff B006 (mutable defaults, opt-in B), Pylint W0102

SPECIMEN GALLERY

REAL PATTERNS. REAL PROBLEMS.

Mutable default argument — B006

```
def add_item(item, items=[]):
    items.append(item)
    return items
```

The same list is shared across calls, leading to unexpected state.

Closure late binding

```
funcs = []
for i in range(3):
    funcs.append(lambda: i)
# All functions see the last i.
```

The loop variable is looked up when the function runs, not when it's defined.

Bare except — E722

```
try:
    risky_operation()
except:
    pass
```

Swallows KeyboardInterrupt and SystemExit, hiding important exceptions.

List multiplication aliasing — W0102

```
matrix = [[0]] * n
matrix[0][0] = 1
# All rows reference the same list.
```

Creates multiple references to the same inner list, causing surprising updates.

> Spot the patterns. Write more reliable Python.

Correctness, Reliability, and Readability

CLEAN CODE
RELIABLE SYSTEMS
MAINTAINABLE FUTURES

Correctness: specific exception types, context managers for files, connections, and locks

Minimal try-block scope and deterministic behavior over happy-path assumptions

Readability: descriptive names, single-responsibility functions, shallow nesting, no magic numbers

Structure: light `__init__`, dataclasses for pure data, clear boundaries, backward-compatible interfaces

Heuristic targets: cyclomatic complexity ≤ 10, functions around 40–60 lines

Python 3.15 (October 2026) raises the bar: Python 3.10 reached end of life October 1, 2026

```
1  def process_data(path):
2      try:
3          with open(path) as f:
4              data = f.read()
5          result = parse(data)
6          if result is None:
7              raise ValueError(
8                  "Invalid data"
19             )
11         return result
11     except FileNotFoundError:
12         raise
14     except ValueError as exc:
15         raise RuntimeError(
16             "Processing failed"
19         ) from exc
17
...
58
59
60
```

Names
Clear, descriptive function name

Nesting
Shallow nesting is easier to read

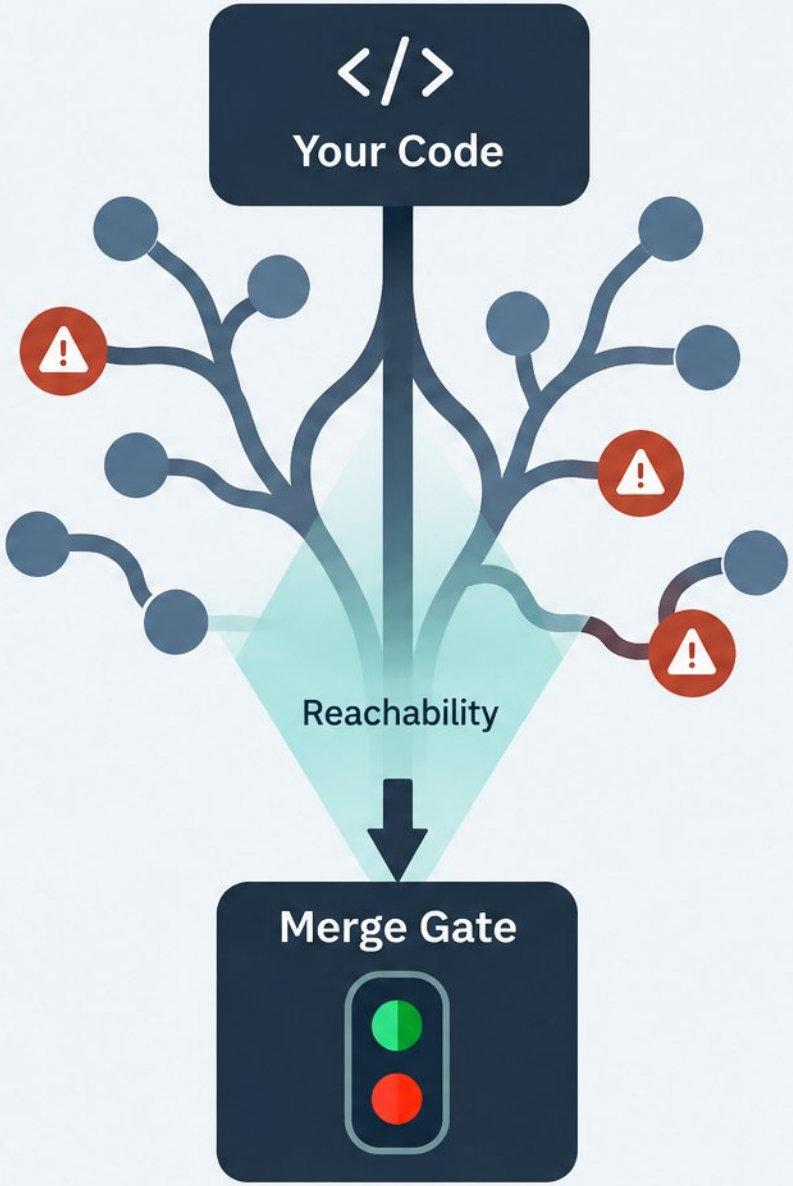
Cyclomatic complexity ≤ 10

Function length
Aim for 40–60 lines

Security and Dependency Hygiene

SAFER CODE • STRONGER SUPPLY CHAIN • HIGHER CONFIDENCE

- **OWASP must-fixes:** parameterized SQL, no `shell=True`, no `pickle.loads`/`yaml.load` on untrusted data
- **Secrets:** nothing hardcoded, use env vars or a secrets manager, `detect-secrets` baseline committed
- **Crypto pitfalls:** no MD5/SHA1 passwords, `secrets` over `random`, no `verify=False`, `defusedxml` for XML
- **Dependencies:** `pip-audit` against the lockfile in CI; HIGH/CRITICAL CVEs block merge
- **Prioritize with reachability:** SAST on your code plus reachability-aware SCA on dependencies



Pythonic Style: PEP 8, PEP 20, and Project Conventions

CLEANER CODE. CLEARER THINKING.

PEP 8 Consistency Hierarchy

Guide

Project

Module or function

When to Deviate

- Readability
- Surrounding code
- Legacy code

Zen of Python and Docstrings

"The Zen of Python as a design filter."

PEP 257 for docstrings

Formatter and Import Sorting

```
Black and Ruff
format default to
88 characters;
ignore E203.
```

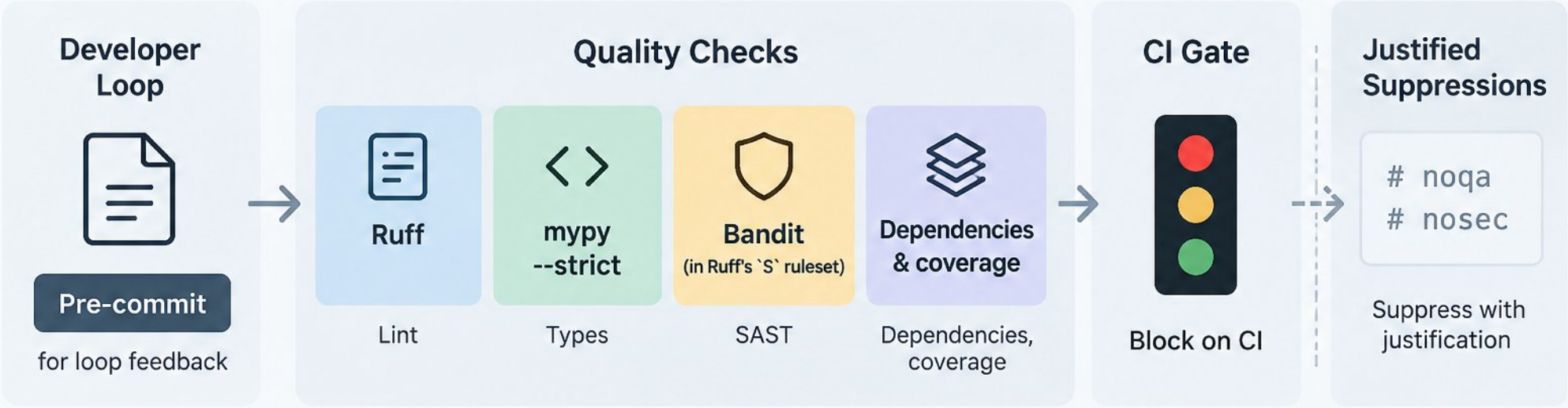
Ruff I sorts three import groups.



- PEP 8 consistency hierarchy: guide < project < module or function
- Deviate for readability, surrounding code, or legacy code
- Zen of Python as a design filter; PEP 257 for docstrings
- Black and Ruff format default to 88 characters; ignore E203
- Conventions live in pyproject.toml; Ruff I sorts three import groups

Static Analysis and Automated Quality Gates

- Ruff in 2026: 900+ lint rules, 0.2 s on 50k lines vs ~20 s for Flake8
- mypy --strict as the trusted CI reference; Pyright faster in editors
- Match tool to job: lint, types, SAST, dependencies, coverage
- Fold Bandit checks into Ruff's `S` ruleset for one fast pass
- Pre-commit for loop feedback; block on CI, suppress with justification



CLEANER CODE | SAFER SOFTWARE | HIGHER CONFIDENCE

Testing, Coverage, and Runtime Signals

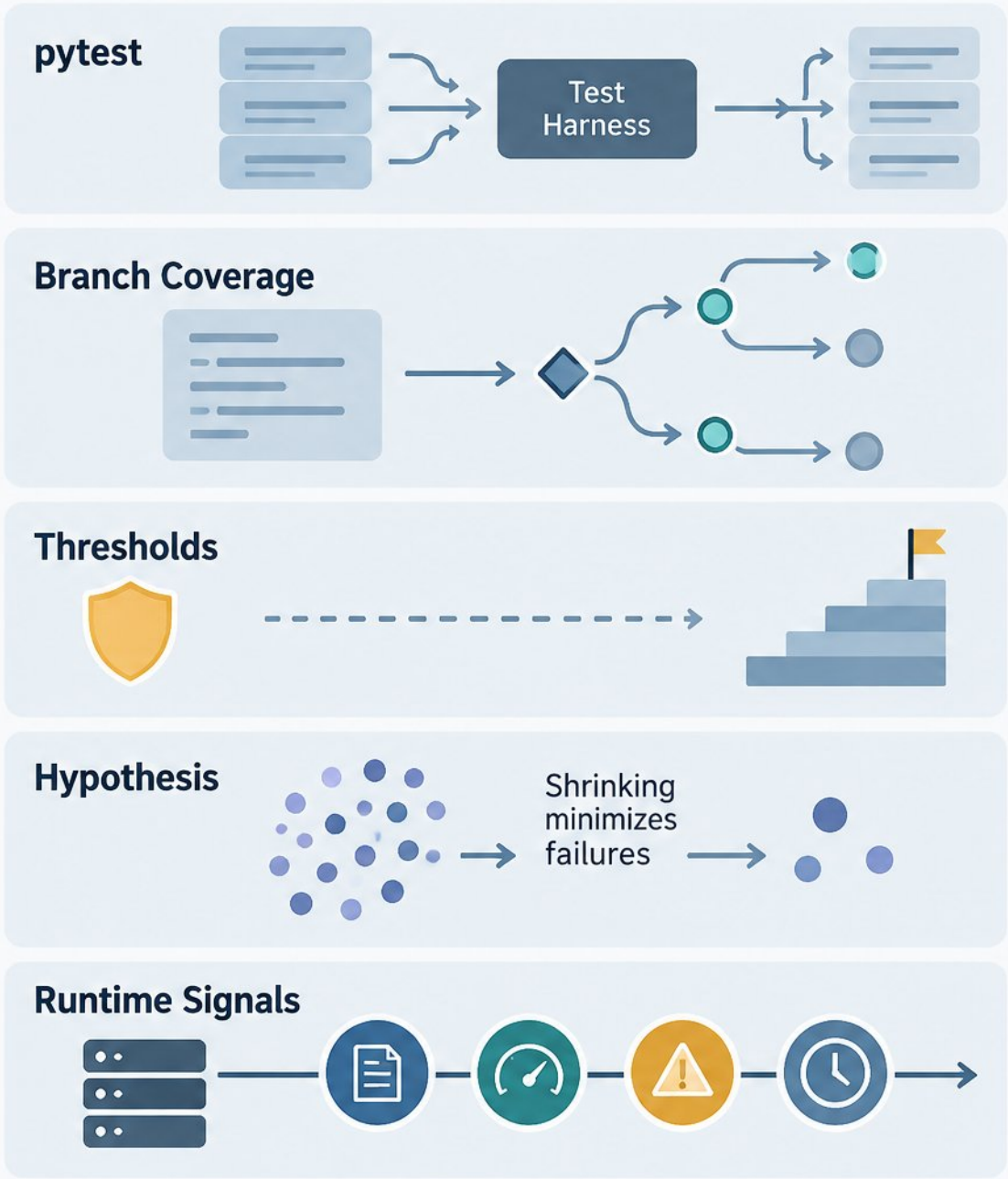
pytest default: parametrize variations, narrow fixtures, deterministic tests, no shared mutable state

Branch coverage over line coverage: `branch = true`; the percentage drops because the question got sharper

Thresholds: ~80% overall CI floor, 100% on critical paths; coverage is a floor, not a goal

Hypothesis: property-based tests find ~50x the mutations of an average unit test; shrinking minimizes failures

Runtime signals: structured logging without PII, latency/error metrics, timeouts on external calls, profile first



Code Review Practices for Python Teams

Better code. Stronger teams.



- Google's standard: approve once the change definitely improves code health, not perfection
- Reviewers check design, functionality, complexity, tests, naming, comments, style, docs
- Respond within one business day; explain the why; prefix nits with "Nit:"
- Authors: small PRs (under 400 LOC), clear why/what/how-to-test, self-review, passing tests
- If a linter can decide it, it is not a review comment: add or fix the gate

CLEANER CODE. BRIGHTER OUTCOMES.

A Practical Python Review Ledger

- Work the checklist like a ledger: PASS, FAIL, or N/A, with `file:line` evidence
- Comments come only from FAIL rows, one per finding
- A green pipeline is not a clean change; real PRs still yield FAIL rows
- Severity-first sweep: correctness, Python pitfalls, concurrency, security, performance
- Merge blockers: no `cast()`, no module-level mutable state, no `except Exception`, one name per concept

A green pipeline is not a clean change; real PRs still yield FAIL rows

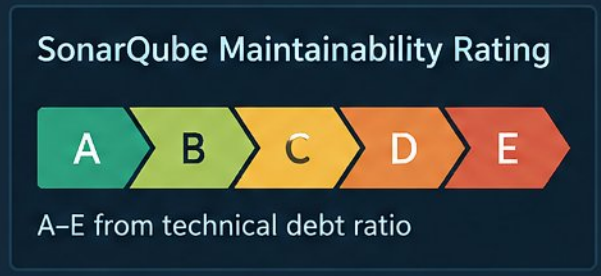
CHECK	RESULT	EVIDENCE	COMMENT
	PASS	—	—
	FAIL	file:line	
	N/A	—	—
	FAIL	file:line	
	PASS	—	—

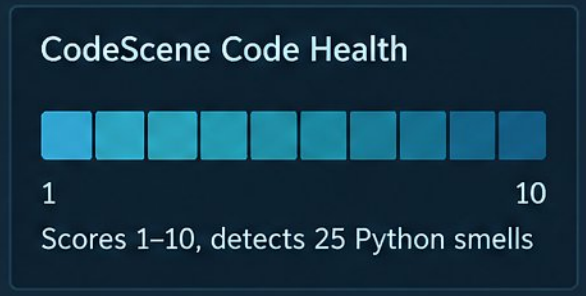
Measuring Quality Without Gaming the Metrics

BETTER CODE HIGHER CONFIDENCE LASTING IMPACT

- Four metric families: static, test, review, and runtime
- SonarQube rates maintainability A–E from technical debt ratio
- CodeScene Code Health scores 1–10, detects 25 Python smells
- Track trends and baselines in frequently edited code
- Code metrics beat design metrics for fault prediction
- Relative churn beats absolute; no single metric wins
- Goodhart's law: never tie metrics to individual performance









Goodhart's law: never tie metrics to individual performance

MEASURE LEARN IMPROVE

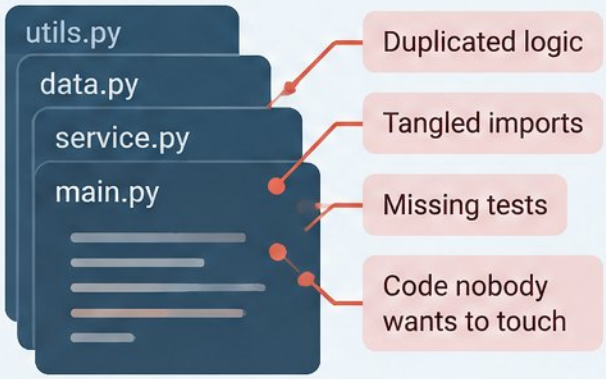
CODE QUALITY BUILDS TOMORROW

Technical Debt, Refactoring, and Dependency Management

Cleaner Code
Healthier Projects
More Sustainable Outcomes

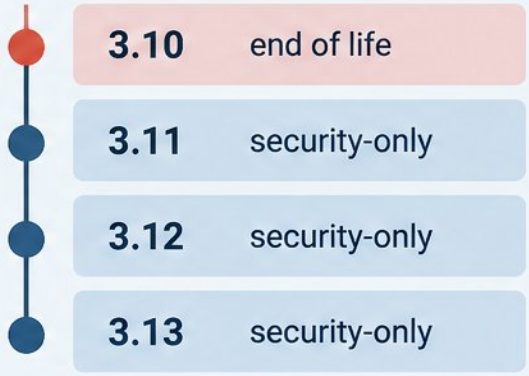
1. Debt signals

- Debt signals: duplicated logic, tangled imports, missing tests, code nobody wants to touch



2. Python version debt

- Python version debt as of October 2026: 3.10 end of life; 3.11–3.13 security-only



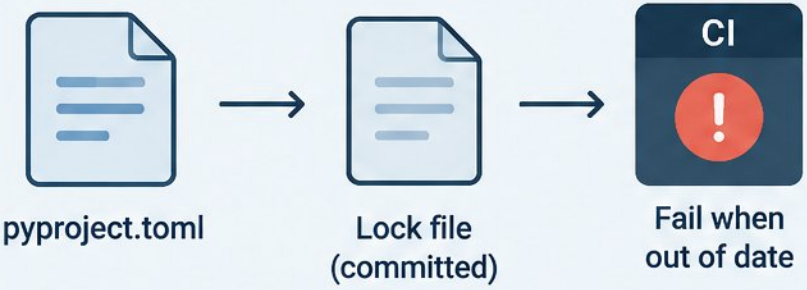
3. Refactor safely

- Refactor safely: characterization tests first, incremental steps, feature flags



4. Lock files

- Lock files: declare ranges in `pyproject.toml`, commit the lock, fail CI when out of date



5. 2026 tools

- 2026 tools: uv (universal `uv.lock`), Poetry 2.x `[project]`, pip-tools now under PyPA; PEP 751 `pylock.toml` partial



Teaching and Scaling Quality Practices

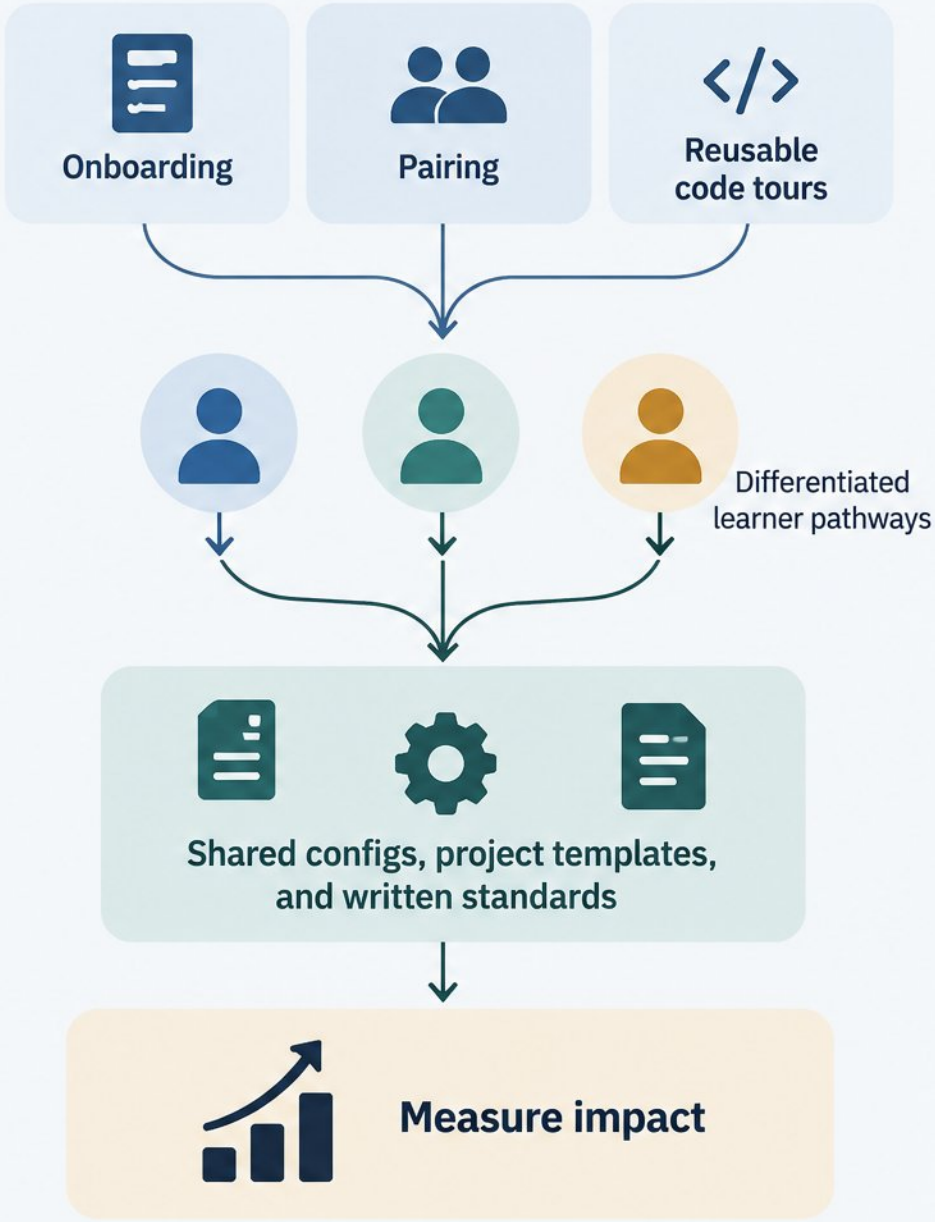
Onboarding: starter checklists, pairing, reusable code tours; expert tours scored 83% vs. 57% AI-only

Align training with the team's own repository: 89% completion vs. 51% for generic content

Gains are uneven: short interventions help less-prepared learners most; add differentiated pathways

Scale via shared Ruff/mypy configs, project templates, and written org-wide standards

Measure impact: before/after static analysis and review metrics; expect recurring finding categories



Capstone: Auditing a Real Python Codebase

READ
ANALYZE
IMPROVE
REPEAT

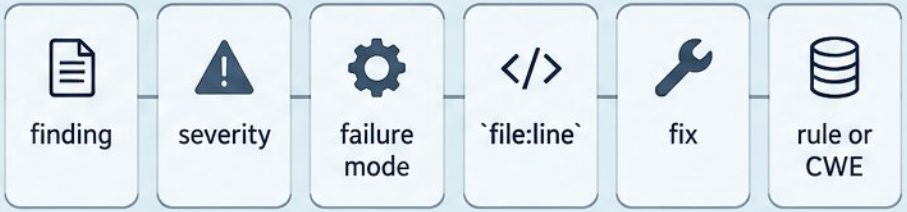
- **Audit workflow:** inventory → automated scans → manual review → risk ranking
- **Automated pass:** Ruff, mypy, Bandit, pip-audit, pytest-cov on the resolved lockfile
- **A score without a justification is a bug:** every score needs evidence
- **Report anatomy:** finding, severity, failure mode, `file:line`, fix, rule or CWE
- **Exercise:** run the review ledger on a sample repo; defend your priorities
- **Next steps:** a roadmap with baselines, targets, and an owner per workstream



Audit Workflow



Report Anatomy



Roadmap



PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/51c2de84/public