🐍 Il Nuovo Paradigma dello Sviluppatore AI

Python Edition - L'ecosistema nativo per AI

v2.0 - Production Ready Edition

Dicembre 2025

📌 Introduzione

Il Messaggio di Andrej Karpathy

"Non mi sono mai sentito così indietro come programmatore"

In un post del 26 dicembre 2025, Andrej Karpathy ha descritto l'AI come **"tecnologia aliena senza manuale"**. E Python è il linguaggio con cui questa "tecnologia aliena" parla nativamente.

Il contesto Python-AI

Python non è solo il linguaggio dominante per AI/ML - è l'**ecosistema nativo** in cui l'intera rivoluzione AI sta accadendo. Ogni major LLM SDK, framework, e tool è Python-first.

ℹ️ Questa Guida Python Edition

Versione specializzata per sviluppatori Python che include:

- Setup con Poetry, uv, pyenv
- Type hints e mypy integration
- pytest e testing moderno
- Ruff per linting velocissimo
- FastAPI + SQLAlchemy patterns
- LangChain, CrewAI, e AI frameworks

- Security per Python (bandit, safety)

🐍 Perché Python Governa l'AI

I Numeri Parlano Chiaro

Ecosistema AI/ML

- **99%** dei paper ML usa Python
- **PyTorch, TensorFlow, JAX** - tutti Python-first
- **Hugging Face** - ecosystem completamente Python
- **LangChain, LlamaIndex** - nati in Python
- **OpenAI, Anthropic SDKs** - Python è il tier 1

Vantaggi per AI Development

Aspetto	Python	Altri Linguaggi
SDK AI	✅ Nativi, completi, aggiornati	⚠️ Wrapper, lag features
ML Libraries	✅ PyTorch, TF, scikit-learn	❌ Limitati o assenti
Community	✅ Enorme, attiva, bleeding-edge	⚠️ Più piccole
Prototyping	✅ Velocissimo, REPL	⚠️ Più verbose
Notebooks	✅ Jupyter nativo	⚠️ Supporto limitato
Type Safety	⚠️ Opzionale (mypy)	✅ Spesso nativa
Performance	⚠️ Più lento (ma NumPy/C bindings)	✅ Più veloce

✅ Il Verdetto

Per AI/ML development, Python è **indispensabile**. Anche se conosci altri linguaggi, Python è il tuo passaporto per l'ecosistema AI.

Python Moderno (2025)

Il Python di oggi è molto diverso da quello di 5-10 anni fa:

- **Type hints** obbligatori (PEP 484+)
- **Pattern matching** (Python 3.10+)
- **Async/await** maturo e diffuso
- **Dataclasses e Pydantic** per data validation
- **uv e rye** - package manager velocissimi
- **Ruff** - linter 100x più veloce di pylint

🛠️ Tool per Python AI Development

IDE & Coding Assistants

VSCode + Continue

Scelta raccomandata per iniziare

✅ PRO

- Python extension Microsoft eccellente
- Jupyter notebooks integrati
- Continue gratis e open-source
- Debugging top-tier
- Remote development (SSH, containers)

❌ CONTRO

- Configurazione iniziale più complessa
- Meno "magic" di PyCharm

PyCharm Professional + AI Assistant

La suite completa enterprise

> **Pricing:** $249/anno professional, free per studenti/open-source

✅ PRO

- Refactoring Python migliore in assoluto
- Database tools integrati
- Scientific mode (Jupyter, NumPy)
- Django/Flask/ FastAPI support nativo
- AI Assistant sempre più potente

❌ CONTRO

- Pesante su risorse
- A pagamento
- AI integration ancora dietro a Cursor

Cursor

AI-first, Python supported

Stesso discorso della versione Node.js - Composer mode eccellente, ma fork di VSCode quindi potrebbe essere dietro su feature Python-specific.

Python-Specific AI Tools

Aider - Python Native

Aider è scritto in Python e ha supporto eccellente per ecosistema Python.

- Capisce virtual environments
- Integrazione con pytest

- Type hints aware
- Poetry/pip/pipenv support

GitHub Copilot

Eccellente per Python:

- Addestrato su moltissimo codice Python
- Buono con pandas, numpy, requests
- Suggerimenti type hints

Notebooks AI-Enhanced

Jupyter + Copilot

VSCode Jupyter + GitHub Copilot = combo perfetta per data exploration

Google Colab + Gemini

Free GPU + AI assistant integrato

Cursor Notebook Mode

Jupyter notebooks con Composer - interessante per ML prototyping

⚙️ Setup Python Moderno su Mac

1️⃣ Python Version Management

Opzione A: pyenv (Raccomandato)

```
# Installa pyenv
brew install pyenv

# Setup shell (zsh)
echo 'export PYENV_ROOT="$HOME/.pyenv"' >> ~/.zshrc
echo 'command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"' >>
echo 'eval "$(pyenv init -)"' >> ~/.zshrc

# Reload shell
source ~/.zshrc

# Installa Python 3.12 (LTS attuale)
pyenv install 3.12

# Set come default
pyenv global 3.12

# Verifica
python --version  # deve essere 3.12.x
```

Opzione B: uv (Nuovo, Velocissimo)

```
# uv è un package manager Python scritto in Rust - 10-100x più veloce
brew install uv
```

```
# Installa Python
uv python install 3.12

# Crea virtual environment
uv venv

# Attiva
source .venv/bin/activate
```

Python Versions (Dic 2025)

Raccomandato: Python 3.12 (current stable)

Minimum: Python 3.10 (per pattern matching e nuove features)

Python 3.13: in beta, non per produzione ancora

2 Package Management

Poetry (Opzione Moderna)

```
# Installa Poetry
curl -sSL https://install.python-poetry.org | python3 -

# Aggiungi a PATH
echo 'export PATH="$HOME/.local/bin:$PATH"' >> ~/.zshrc
source ~/.zshrc

# Verifica
poetry --version
```

Setup nuovo progetto con Poetry

```
# Crea nuovo progetto
poetry new my-ai-project
cd my-ai-project

# Oppure in progetto esistente
poetry init

# Aggiungi dependencies
poetry add anthropic openai langchain fastapi

# Dev dependencies
poetry add --group dev pytest ruff mypy black

# Attiva virtual environment
poetry shell
```

pyproject.toml esempio

```
[tool.poetry]
name = "my-ai-project"
version = "0.1.0"
description = "AI-powered application"
authors = ["Your Name "]
readme = "README.md"
python = "^3.12"

[tool.poetry.dependencies]
python = "^3.12"
anthropic = "^0.40.0"
fastapi = "^0.115.0"
uvicorn = "^0.32.0"
pydantic = "^2.10.0"
sqlalchemy = "^2.0.0"
```

```
[tool.poetry.group.dev.dependencies]
pytest = "^8.3.0"
ruff = "^0.8.0"
mypy = "^1.13.0"
black = "^24.10.0"
pytest-cov = "^6.0.0"
pytest-asyncio = "^0.24.0"

[build-system]
requires = ["poetry-core"]
build-backend = "poetry.core.masonry.api"

[tool.ruff]
line-length = 100
target-version = "py312"

[tool.mypy]
python_version = "3.12"
strict = true
warn_return_any = true
warn_unused_configs = true

[tool.pytest.ini_options]
testpaths = ["tests"]
python_files = ["test_*.py"]
python_classes = ["Test*"]
python_functions = ["test_*"]
addopts = "-v --cov=src --cov-report=html"
```

3 VSCode Setup per Python

```
# Estensioni essenziali
code --install-extension ms-python.python
code --install-extension ms-python.vscode-pylance
```

```
code --install-extension continue.continue
code --install-extension charliermarsh.ruff
code --install-extension ms-toolsai.jupyter

# Opzionali ma utili
code --install-extension GitHub.copilot
code --install-extension GitHub.copilot-chat
code --install-extension tamasfe.even-better-toml
```

VSCode settings.json per Python

```
{
  // Python
  "python.defaultInterpreterPath": "${workspaceFolder}/.venv/bin/python",
  "python.terminal.activateEnvironment": true,

  // Linting con Ruff (velocissimo)
  "ruff.enable": true,
  "ruff.lint.run": "onSave",
  "ruff.format.args": ["--line-length", "100"],

  // Type checking
  "python.analysis.typeCheckingMode": "basic",
  "python.analysis.autoImportCompletions": true,

  // Testing
  "python.testing.pytestEnabled": true,
  "python.testing.unittestEnabled": false,

  // Formatting
  "editor.formatOnSave": true,
  "[python]": {
    "editor.defaultFormatter": "charliermarsh.ruff",
    "editor.codeActionsOnSave": {
      "source.organizeImports": "explicit"
    }
```

```
  }
}
```

4 Aider per Python

```
# Già installato? Usa quello
aider --version

# Oppure installa con uv (più veloce)
uv tool install aider-chat

# Config per Python
touch ~/.aider.conf.yml
```

~/.aider.conf.yml per Python

```
model: claude-sonnet-4-20250514
dark-mode: true
pretty: true
stream: true
auto-commits: true
dirty-commits: false
attribute-commits: true
edit-format: diff

# Python specific
lint-cmd: "ruff check ."
test-cmd: "pytest"
auto-lint: true
auto-test: false  # true se vuoi test automatici
```

5 Configurazione Progetto

.cursorrules (o .continuerules) per Python

```
# .cursorrules

You are an expert Python developer working with modern Python 3.12+.

Tech stack:
- Python 3.12+
- FastAPI for APIs
- SQLAlchemy 2.0 for ORM
- Pydantic v2 for validation
- pytest for testing
- Type hints everywhere

Code style:
- ALWAYS use type hints (mypy strict mode)
- Use dataclasses or Pydantic models
- Prefer async/await for I/O operations
- Follow PEP 8 (enforced by ruff)
- Max line length: 100
- Use descriptive variable names

When writing code:
- ALWAYS add type hints to function signatures
- Use Pydantic for data validation
- Handle errors with proper exception types
- Use context managers (with statements)
- Prefer pathlib over os.path
- Use f-strings for formatting
- NEVER use mutable default arguments

Security:
- Never commit API keys or secrets
- Validate all inputs with Pydantic
```

```
- Use parameterized queries with SQLAlchemy
- Follow OWASP Python Security
- Use secrets module for random values

Testing:
- Use pytest for all tests
- Write tests using arrange-act-assert pattern
- Use fixtures for setup
- Test edge cases and error paths
- Aim for >80% coverage

AI/ML specific:
- Use type hints even for tensors/arrays
- Document model architectures
- Version datasets and models
- Log hyperparameters
- Handle GPU/CPU fallback

Before committing:
- Run: ruff check . && ruff format .
- Run: mypy .
- Run: pytest
- Verify no secrets
- Check diff size (<300 lines)
```

6 Environment Variables

```
# .env file
ANTHROPIC_API_KEY=your_key_here
OPENAI_API_KEY=your_key_here
DATABASE_URL=postgresql://localhost/mydb
ENVIRONMENT=development

# .env.example (commit questo)
ANTHROPIC_API_KEY=sk-ant-xxxxx
```

```
OPENAI_API_KEY=sk-xxxxx
DATABASE_URL=postgresql://localhost/mydb
ENVIRONMENT=development
```

Carica con python-dotenv

```
# poetry add python-dotenv

# In main.py o config.py
from dotenv import load_dotenv
import os

load_dotenv()

ANTHROPIC_API_KEY = os.getenv("ANTHROPIC_API_KEY")
if not ANTHROPIC_API_KEY:
    raise ValueError("ANTHROPIC_API_KEY not set")
```

7 Verifica Setup

```
# Python version
python --version  # >= 3.12

# Poetry
poetry --version

# Virtual env attivo?
which python  # deve puntare a .venv

# Packages installati?
poetry show

# Ruff funziona?
ruff check .
```

```
# mypy funziona?
mypy --version

# pytest funziona?
pytest --version
```

✅ Setup Python Completo!

Ora hai un ambiente Python moderno, type-safe, con AI tools integrati.

🔒 Security Python-Specific

1. Secrets Management

.gitignore per Python

```
# .gitignore

# Secrets
.env
.env.local
*.key
*.pem
config/secrets.py

# Virtual environments
.venv/
venv/
env/
ENV/

# Python
__pycache__/
*.py[cod]
*$py.class
*.so
.Python
build/
dist/
*.egg-info/

# AI tools
.aider*
.cursor/
.continue/
```

```
# IDE
.vscode/
.idea/
*.swp

# Testing
.pytest_cache/
.coverage
htmlcov/
.tox/

# Jupyter
.ipynb_checkpoints/
*.ipynb  # se non vuoi committare notebooks
```

2. Security Scanning Tools

```
# Installa security tools
poetry add --group dev bandit safety pip-audit

# Bandit - scan per vulnerabilità
bandit -r src/

# Safety - check dependencies vulnerabili
safety check

# pip-audit - audit dependencies
pip-audit
```

Integra in pre-commit

```
# .pre-commit-config.yaml
repos:
```

```
  - repo: https://github.com/PyCQA/bandit
    rev: '1.7.10'
    hooks:
      - id: bandit
        args: ['-c', 'pyproject.toml']

  - repo: https://github.com/charliermarsh/ruff-pre-commit
    rev: 'v0.8.0'
    hooks:
      - id: ruff
      - id: ruff-format
```

3. Input Validation con Pydantic

```
from pydantic import BaseModel, EmailStr, Field, validator
from typing import Optional

class UserCreate(BaseModel):
    email: EmailStr  # valida email automaticamente
    username: str = Field(..., min_length=3, max_length=50)
    age: Optional[int] = Field(None, ge=0, le=150)

    @validator('username')
    def username_alphanumeric(cls, v):
        if not v.isalnum():
            raise ValueError('must be alphanumeric')
        return v

# FastAPI endpoint
from fastapi import FastAPI, HTTPException

app = FastAPI()

@app.post("/users")
async def create_user(user: UserCreate):
```

```
    # Pydantic ha già validato tutto
    # Se arriviamo qui, i dati sono sicuri
    return {"email": user.email}
```

4. SQL Injection Prevention

```
# ❌ MAI COSÌ (SQL injection vulnerability)
query = f"SELECT * FROM users WHERE email = '{email}'"
db.execute(query)

# ✅ SEMPRE COSÌ (SQLAlchemy ORM)
from sqlalchemy import select
stmt = select(User).where(User.email == email)
result = await session.execute(stmt)

# ✅ OPPURE con parametri (raw SQL)
query = "SELECT * FROM users WHERE email = :email"
result = db.execute(query, {"email": email})
```

5. Secure Random Values

```
# ❌ MAI random module per security
import random
token = random.randint(1000, 9999)  # INSICURO!

# ✅ SEMPRE secrets module
import secrets
token = secrets.token_urlsafe(32)
api_key = secrets.token_hex(32)
```

🚫 Python Security Checklist

- ☐ No eval(), exec(), o __import__ dinamici
- ☐ No pickle da fonti non fidate
- ☐ Valida sempre con Pydantic
- ☐ Use secrets, non random per crypto
- ☐ Parametrizza SQL queries
- ☐ No shell=True in subprocess
- ☐ Sanitizza file uploads

🧪 Testing Python Moderno

1. Setup pytest

```
# poetry add --group dev pytest pytest-cov pytest-asyncio pytest-mock

# pyproject.toml
[tool.pytest.ini_options]
testpaths = ["tests"]
python_files = ["test_*.py", "*_test.py"]
python_classes = ["Test*"]
python_functions = ["test_*"]
addopts = [
    "-v",
    "--cov=src",
    "--cov-report=html",
    "--cov-report=term-missing",
    "--cov-fail-under=80"
]
asyncio_mode = "auto"
```

2. Test Structure

```
# tests/test_user_service.py

import pytest
from src.services.user import UserService
from src.models import User

@pytest.fixture
async def user_service():
    """Setup user service for tests."""
    service = UserService()
```

```
    await service.initialize()
    yield service
    await service.cleanup()

@pytest.fixture
def sample_user_data():
    """Sample user data for testing."""
    return {
        "email": "test@example.com",
        "username": "testuser",
        "age": 25
    }

class TestUserService:
    """Tests for UserService."""

    async def test_create_user_success(
        self,
        user_service: UserService,
        sample_user_data: dict
    ):
        """Test successful user creation."""
        # Arrange
        email = sample_user_data["email"]

        # Act
        user = await user_service.create_user(**sample_user_data)

        # Assert
        assert user.email == email
        assert user.id is not None
        assert user.created_at is not None

    async def test_create_user_duplicate_email(
        self,
        user_service: UserService,
        sample_user_data: dict
```

```
    ):
        """Test creating user with duplicate email fails."""
        # Arrange
        await user_service.create_user(**sample_user_data)

        # Act & Assert
        with pytest.raises(ValueError, match="Email already exists"):
            await user_service.create_user(**sample_user_data)

    @pytest.mark.parametrize("invalid_email", [
        "notanemail",
        "@example.com",
        "test@",
        ""
    ])
    async def test_create_user_invalid_email(
        self,
        user_service: UserService,
        sample_user_data: dict,
        invalid_email: str
    ):
        """Test creating user with invalid email fails."""
        sample_user_data["email"] = invalid_email

        with pytest.raises(ValueError):
            await user_service.create_user(**sample_user_data)
```

3. Testing AI-Generated Code

Property-Based Testing con Hypothesis

```
# poetry add --group dev hypothesis

from hypothesis import given, strategies as st
```

```
@given(st.emails())
def test_email_validation_property(email: str):
    """Property test: all valid emails should be accepted."""
    user_data = {"email": email, "username": "test"}
    # Questo deve sempre passare per email valide
    validate_user(user_data)

@given(st.text(min_size=1, max_size=100))
def test_username_never_crashes(username: str):
    """Property test: any string input shouldn't crash."""
    # Può fallire validation, ma non deve crashare
    try:
        validate_username(username)
    except ValueError:
        pass  # Expected for invalid input
```

4. Snapshot Testing

```
# poetry add --group dev syrupy

def test_api_response_structure(snapshot):
    """Snapshot test to catch unexpected API changes."""
    response = api.get_user(user_id=1)
    assert response == snapshot
```

5. Coverage Requirements

```
# Esegui tests con coverage
pytest --cov=src --cov-report=html

# Apri report
open htmlcov/index.html
```

```
# Fail se coverage < 80%
pytest --cov=src --cov-fail-under=80
```

6. Type Checking come Test

```
# mypy è parte dei test!
# pyproject.toml

[tool.mypy]
python_version = "3.12"
strict = true
warn_return_any = true
warn_unused_configs = true
disallow_untyped_defs = true

# In CI/CD
- name: Type Check
  run: mypy src/
```

⚠️ Testing Rules per AI Code

- Ogni funzione pubblica → unit test
- Ogni endpoint API → integration test
- Type hints → mypy strict mode
- Coverage minimo 80%
- Property tests per logica complessa
- Snapshot tests per output stabili

🔄 Workflow Python + AI

Scenario 1: Data Science / ML Prototyping

Jupyter + Copilot

```
# In VSCode
# 1. Apri .ipynb file
# 2. GitHub Copilot attivo
# 3. Type hints anche nei notebook!

import pandas as pd
from typing import DataFrame

def clean_data(df: pd.DataFrame) -> pd.DataFrame:
    """Clean and preprocess data."""
    # Copilot suggerisce i passi comuni
    ...
```

Scenario 2: FastAPI Development

Continue + Type Hints

```
# Continue capisce context FastAPI
# Cmd+L: "Create CRUD endpoints for User model"

from fastapi import FastAPI, Depends, HTTPException
from sqlalchemy.ext.asyncio import AsyncSession
```

```
from typing import List

app = FastAPI()

@app.get("/users", response_model=List[UserResponse])
async def get_users(
    skip: int = 0,
    limit: int = 100,
    db: AsyncSession = Depends(get_db)
) -> List[UserResponse]:
    """Get list of users."""
    # AI genera con type hints corretti
    ...
```

Scenario 3: LangChain Development

Aider per Agent Chains

```
aider

# "Create a LangChain agent that:
# - Uses Claude Sonnet 4
# - Has tools for web search and calculations
# - Maintains conversation history
# - Uses Pydantic for structured output"

# Aider genera tutto con type hints e tests
```

Tool Selection Matrix (Python)

Task	Best Tool	Note
Jupyter notebook exploration	VSCode + Copilot	Native Jupyter support
FastAPI endpoint	Continue Cmd+I	Capisce Pydantic models
Refactor module	Aider	Mantiene type hints
Write tests	Copilot/Continue	Suggerisce fixtures
Data pipeline	Cursor Composer	Multi-file coordination
ML training script	Continue + Jupyter	Iterative development

🤖 AI Frameworks Python

LLM SDKs

Anthropic Python SDK

```
poetry add anthropic

from anthropic import Anthropic

client = Anthropic(api_key=os.getenv("ANTHROPIC_API_KEY"))

message = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[
        {"role": "user", "content": "Hello, Claude!"}
    ]
)

print(message.content[0].text)
```

Orchestration Frameworks

LangChain

```
poetry add langchain langchain-anthropic

from langchain_anthropic import ChatAnthropic
from langchain_core.prompts import ChatPromptTemplate
```

```
llm = ChatAnthropic(
    model="claude-sonnet-4-20250514",
    temperature=0
)

prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful assistant."),
    ("user", "{input}")
])

chain = prompt | llm
response = chain.invoke({"input": "Hello!"})
```

LlamaIndex

Per RAG (Retrieval-Augmented Generation)

```
poetry add llama-index llama-index-llms-anthropic

from llama_index.core import VectorStoreIndex, SimpleDirectoryReader
from llama_index.llms.anthropic import Anthropic

llm = Anthropic(model="claude-sonnet-4-20250514")

documents = SimpleDirectoryReader("data").load_data()
index = VectorStoreIndex.from_documents(documents)

query_engine = index.as_query_engine(llm=llm)
response = query_engine.query("What is this about?")
```

CrewAI

Multi-agent orchestration

```
poetry add crewai crewai-tools

from crewai import Agent, Task, Crew

researcher = Agent(
    role="Researcher",
    goal="Research AI trends",
    llm="claude-sonnet-4-20250514"
)

writer = Agent(
    role="Writer",
    goal="Write about AI",
    llm="claude-sonnet-4-20250514"
)

crew = Crew(agents=[researcher, writer])
result = crew.kickoff()
```

Validation & Structured Output

Instructor

Pydantic models da LLM output

```
poetry add instructor

import instructor
from anthropic import Anthropic
```

```
from pydantic import BaseModel

class User(BaseModel):
    name: str
    age: int
    email: str

client = instructor.from_anthropic(Anthropic())

user = client.messages.create(
    model="claude-sonnet-4-20250514",
    max_tokens=1024,
    messages=[
        {"role": "user", "content": "Extract: John Doe, 30, john@example.c
    ],
    response_model=User
)

print(user.name)  # "John Doe" - type-safe!
```

📋 Python Development Policy

Type Hints Policy

🚫 Type Hints Obbligatori

Mai committare codice senza type hints.

```
# ❌ Rifiutato in PR
def process_data(data):
    return data.upper()

# ✅ Accettato
def process_data(data: str) -> str:
    return data.upper()

# ✅ Per casi complessi
from typing import TypedDict, Optional, List

class UserDict(TypedDict):
    name: str
    age: int
    email: Optional[str]

def get_users() -> List[UserDict]:
    ...
```

Diff Size Limits (Python specific)

Diff Size	Requirement
1-100 righe	Self-review + ruff + mypy + pytest
101-200 righe	+ 1 peer review
201-300 righe	+ 2 peer reviews + security check
300+ righe	Split or senior architect approval

Pre-commit Checklist

- ☐ ruff check . --fix
- ☐ ruff format .
- ☐ mypy . (must pass strict mode)
- ☐ pytest --cov=src --cov-fail-under=80
- ☐ bandit -r src/ (no security issues)
- ☐ No secrets in code
- ☐ All type hints present
- ☐ Docstrings for public functions

CI/CD Pipeline Python

```
# .github/workflows/python-ci.yml
name: Python CI

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
```

```
    strategy:
      matrix:
        python-version: ["3.12"]

    steps:
      - uses: actions/checkout@v4

      - name: Setup Python
        uses: actions/setup-python@v5
        with:
          python-version: ${{ matrix.python-version }}

      - name: Install Poetry
        run: |
          curl -sSL https://install.python-poetry.org | python3
          echo "$HOME/.local/bin" >> $GITHUB_PATH

      - name: Install dependencies
        run: poetry install

      - name: Lint with Ruff
        run: poetry run ruff check .

      - name: Format check
        run: poetry run ruff format --check .

      - name: Type check with mypy
        run: poetry run mypy .

      - name: Test with pytest
        run: poetry run pytest --cov=src --cov-fail-under=80

      - name: Security check
        run: poetry run bandit -r src/
```

📚 Risorse Python + AI

Documentazione Ufficiale

- **Anthropic Python SDK:** GitHub
- **LangChain Python:** python.langchain.com
- **FastAPI:** fastapi.tiangolo.com
- **Pydantic:** docs.pydantic.dev

Tools & Libraries

- **Poetry:** python-poetry.org
- **Ruff:** docs.astral.sh/ruff
- **pytest:** docs.pytest.org
- **mypy:** mypy.readthedocs.io

Learning Resources

- **Python Type Checking:** RealPython guide
- **FastAPI Tutorial:** Official docs
- **LangChain Academy:** Free courses
- **Pydantic Tutorial:** Official tutorial

Community

- **r/Python:** General Python
- **r/MachineLearning:** ML discussions
- **LangChain Discord:** AI framework help
- **FastAPI Discord:** API development

🎯 Conclusioni Python Edition

Perché Python per AI

Python non è solo "un buon linguaggio" per AI - è **il linguaggio**. Ogni breakthrough, ogni nuovo framework, ogni paper di ricerca: tutto nasce in Python.

Il Tuo Stack Python + AI

1. **Fondamenta:** Python 3.12+ + pyenv + Poetry
2. **IDE:** VSCode + Continue (o Cursor)
3. **Quality:** Ruff + mypy + pytest
4. **AI Tools:** Aider + Copilot
5. **Frameworks:** FastAPI + Pydantic + LangChain
6. **Type Safety:** Type hints everywhere, mypy strict

Python Best Practices 2025

- **Type hints:** non negoziabili
- **Pydantic:** per ogni input/output
- **async/await:** per I/O operations
- **Poetry/uv:** non più pip requirements.txt
- **Ruff:** sostituisce black + isort + flake8

- **pytest:** con coverage >80%

📝 Prossimi Passi

1. Setup Python 3.12 + Poetry
2. Clone un progetto di esempio con AI
3. Prova LangChain con Claude
4. Build una FastAPI app AI-powered
5. Sperimenta con Jupyter + Copilot

Versione: 2.0 Python Edition (Dicembre 2025)

🐍 Il Nuovo Paradigma dello Sviluppatore AI - Python Edition

v2.0 - Production Ready

Python: il linguaggio nativo dell'ecosistema AI

Guida creata: Dicembre 2025

Per Giuseppe - Esplorando il futuro AI-first