Streamlining Python Development

A Guide to a Modern Project Setup

PyConDE / PyData 2024, April 22nd

Florian Wilhelm

inovex

Dr. Florian Wilhelm

inovex • HEAD OF DATA SCIENCE

FlorianWilhelm

FlorianWilhelm.info

florian.wilhelm@inovex.de

- Mathematical Modelling
- Modern Data Warehousing & Analytics
- Personalisation & RecSys
- Uncertainty Quantification & Causality
- Python Data Stack
- OSS Contributor & Creator of PyScaffold

inovex

WE ARE INOVEX:

- IT Project Center
- Innovation & Excellence
- Wide Range of Services

25 PyCon Attendees
250 Python Users
500 Tech Heads overall

+ WE'RE HIRING!

inovex

inovex.de

Hamburg
Berlin
Köln
Erlangen
Karlsruhe
Pforzheim
Stuttgart
München

SOFTWARE · DATA & AI · INFRASTRUCTURE

Agenda

1. Introduction:
   a. What makes a good project setup?
   b. How do we achieve it?
2. Streamlined Project Setup:
   a. configuration with pyproject.toml
   b. tooling with hatch, ruff, mypy, pytest, …
3. Conclusion

inovex

Introduction

inovex

What makes a streamlined Python Project Setup?

1. efficient development
2. easy collaboration
3. seamless build & deployment

inovex

Concrete Requirements for those Goals

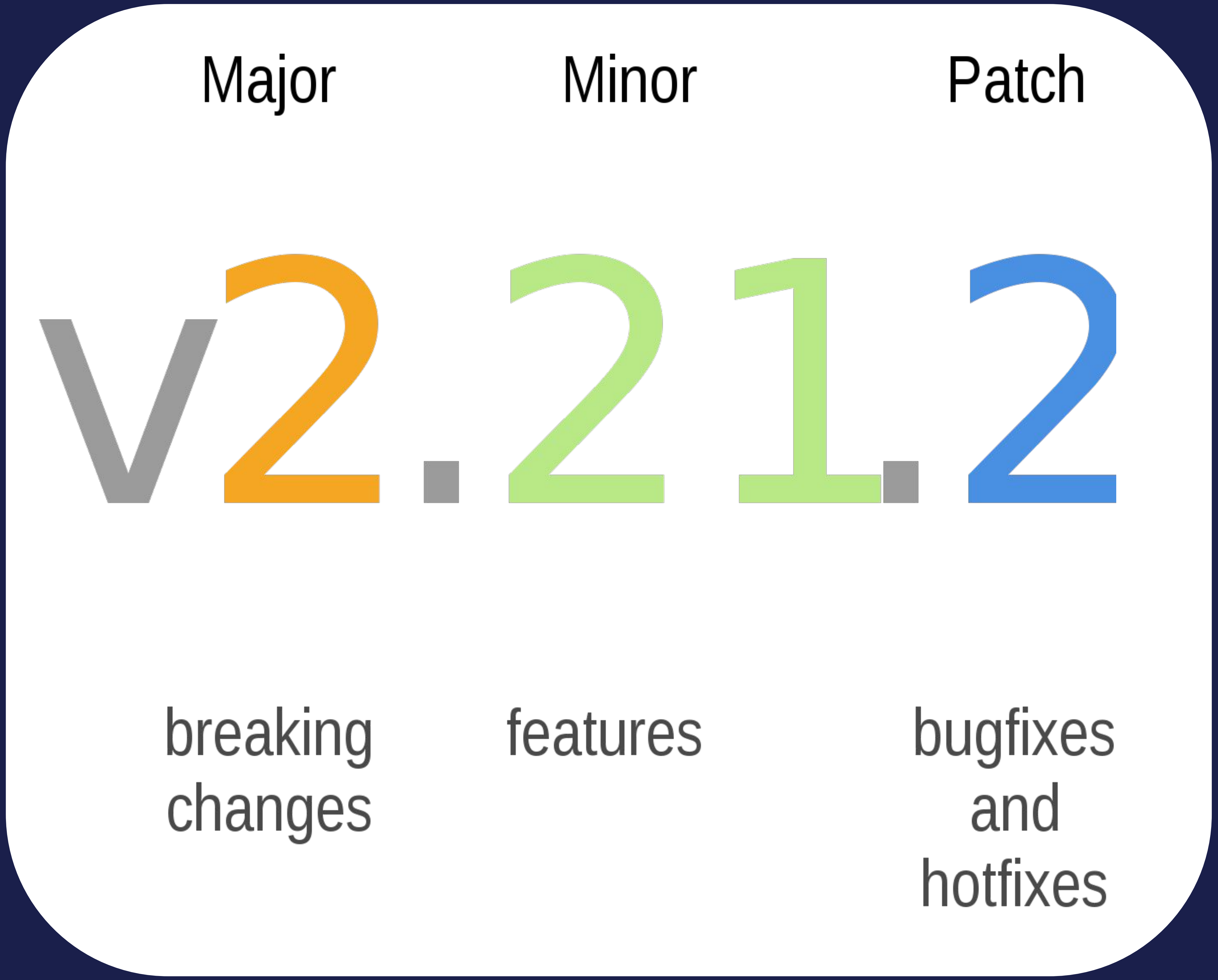
1. Conventions
   a. project structure
   b. code formatting, e.g., pep8, black, ruff
   c. documentation, e.g., Sphinx, mkdocs

2. Automation
   a. dependency & environment management
   b. building & publishing
   c. versioning, e.g., semantic versioning
   d. testing, linting/formatting, type checking

3. Easy to Use!

inovex

Semantic Versioning

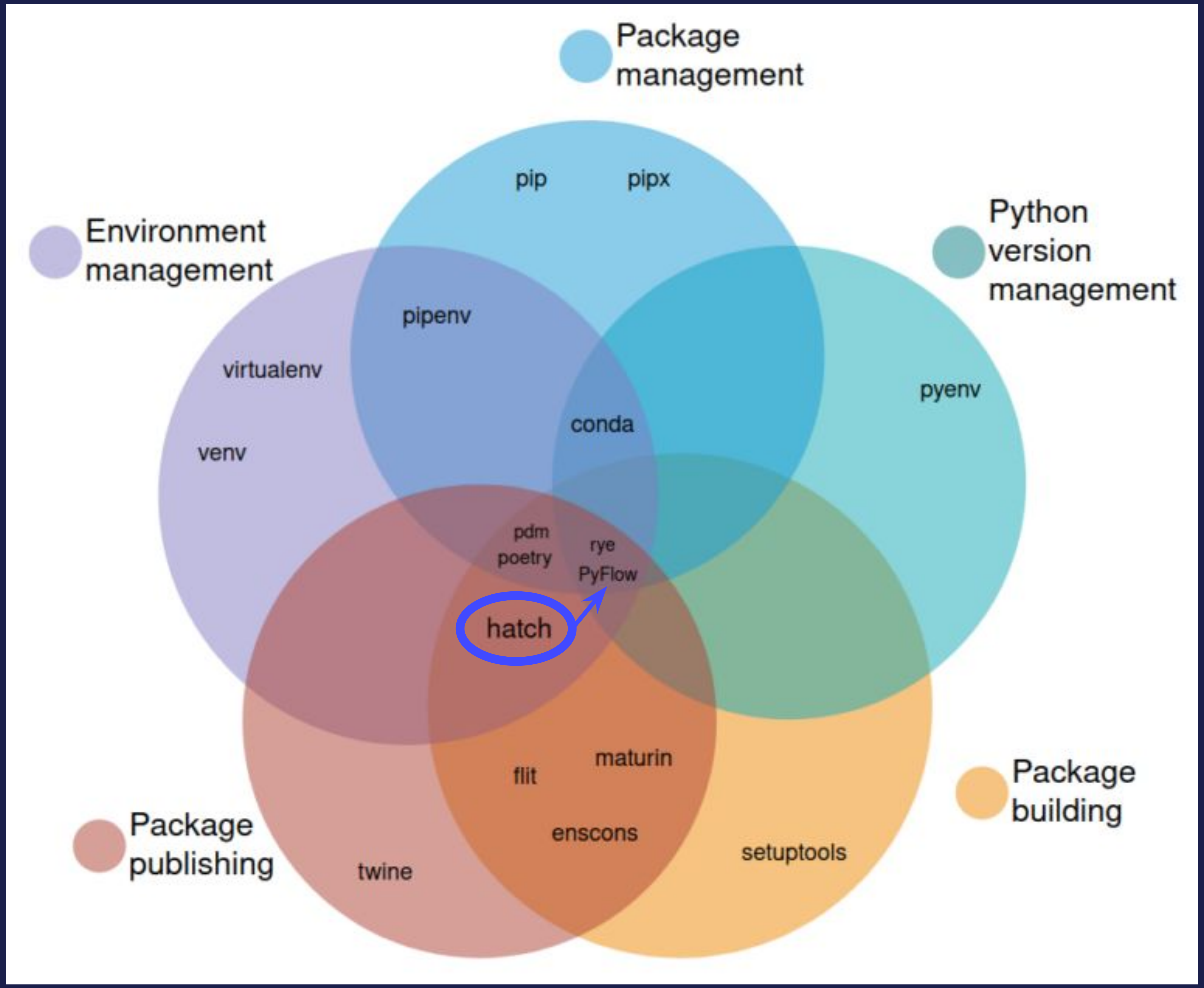


- tells developers what to expect
- avoids dependency hell for developers using your software
- necessary for requirement specifiers like ~= 2.21 or ^2.2.21 (Poetry only)

More Details: https://www.geeksforgeeks.org/introduction-semantic-versioning/ and https://semver.org/

inovex

This is not a talk about the best Package Management Tool



Source: An unbiased evaluation of environment management and packaging tools (https://www.inovex.de/de/blog/)

inovex

Streamlined Project Setup

inovex

🐣 Hatch, the extensible Python project manager

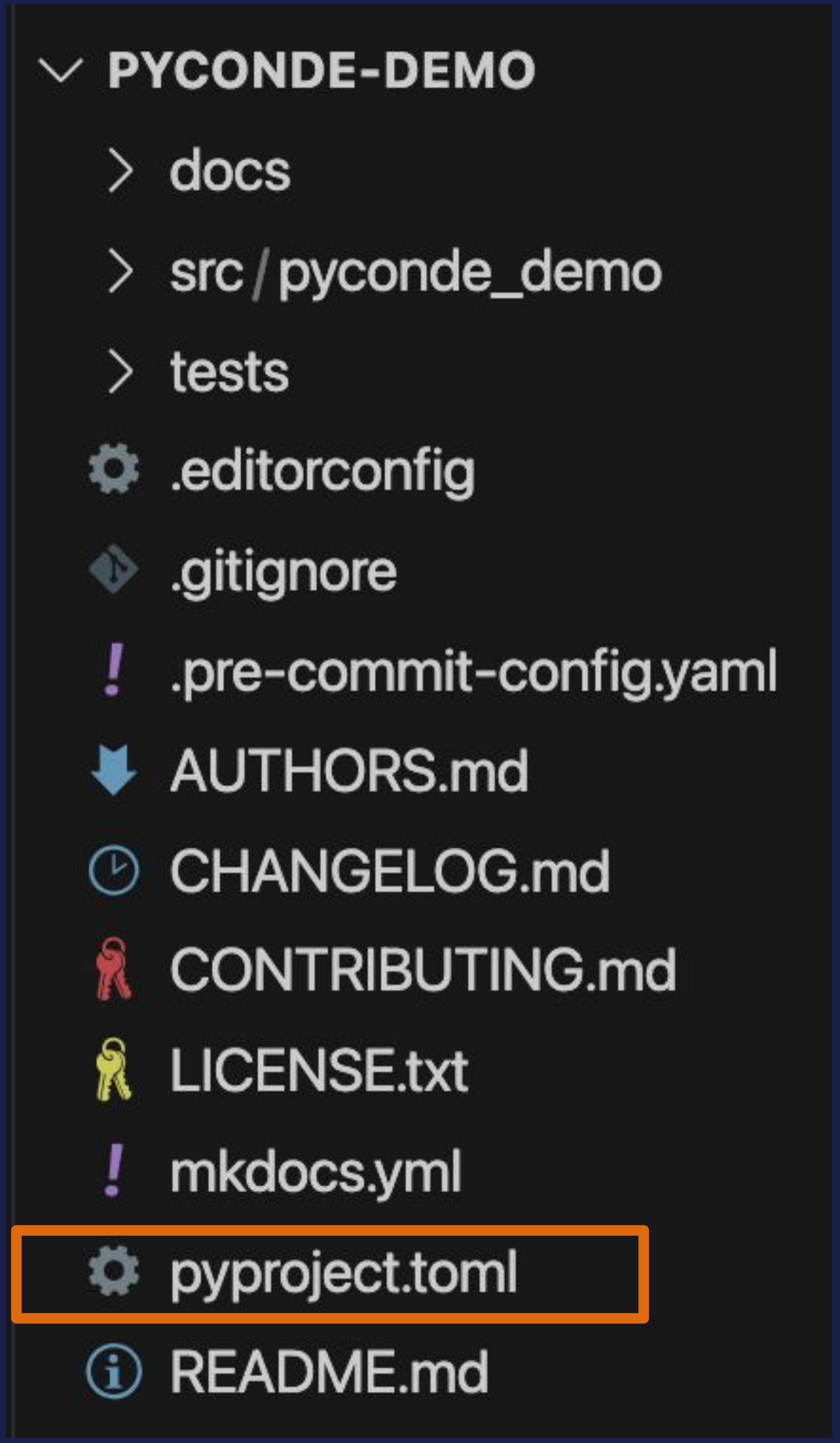
- reproducibly building & publishing packages
- robust environment management with support for custom scripts
- easy Python management, replacing pyenv
- easy semantic versioning based on Git tags
- sophisticated testing within various environments, replacing tox

Ofek Lev

inovex

Project Directory Structure

- folders for
  - source files
  - documentation
  - tests
- human-readable information
  - README.md
  - …
- configuration files
  - pyproject.toml
  - …

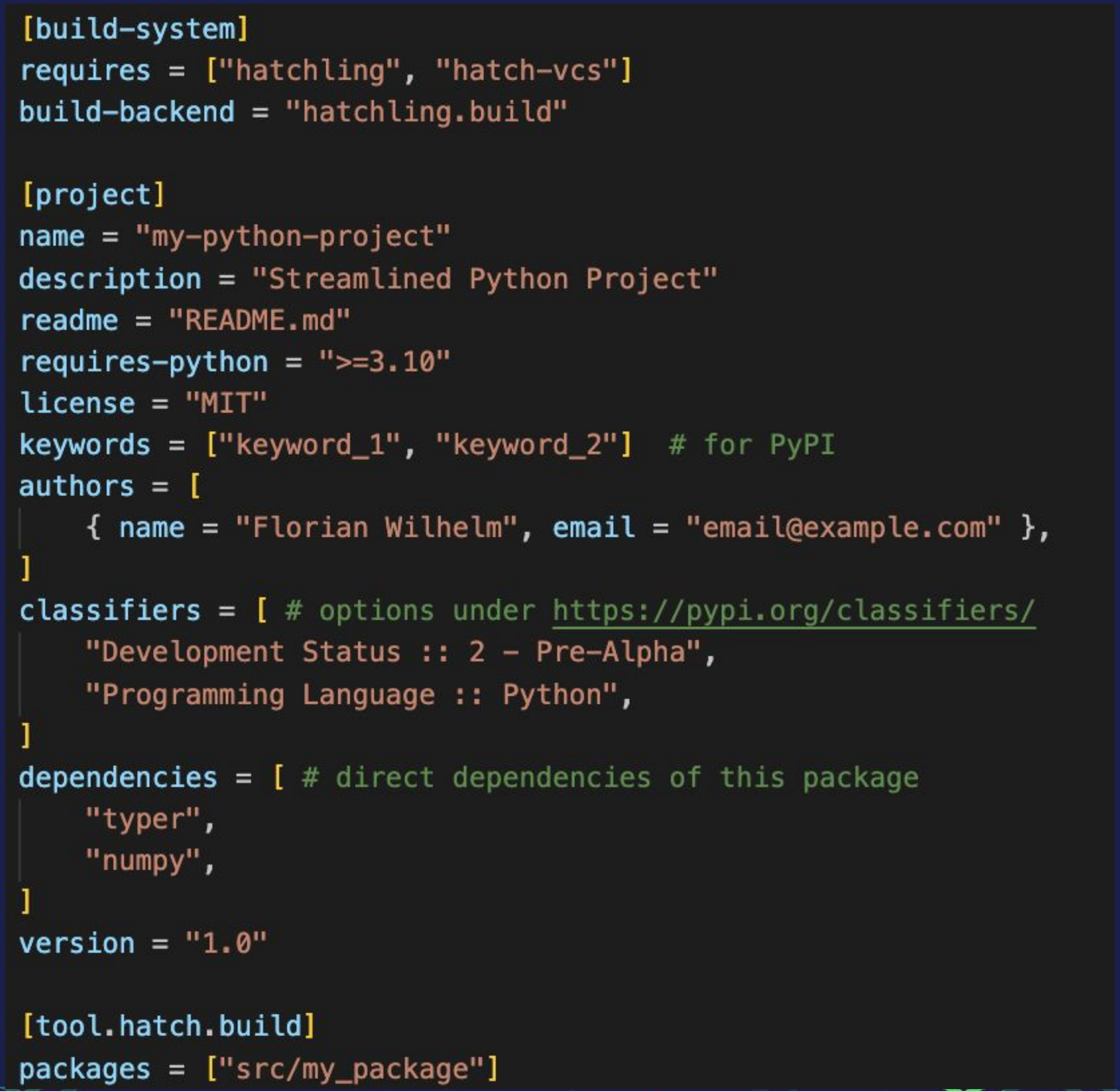


```
PYCONDE-DEMO
  docs
  src / pyconde_demo
  tests
  .editorconfig
  .gitignore
  .pre-commit-config.yaml
  AUTHORS.md
  CHANGELOG.md
  CONTRIBUTING.md
  LICENSE.txt
  mkdocs.yml
  pyproject.toml
  README.md
```

inovex

All-in-One Configuration with pyproject.toml

- defines the build system
- metadata about your project for PyPI
- configuration for (almost) all tools
  - pytest
  - mypy
  - ruff
  - coverage



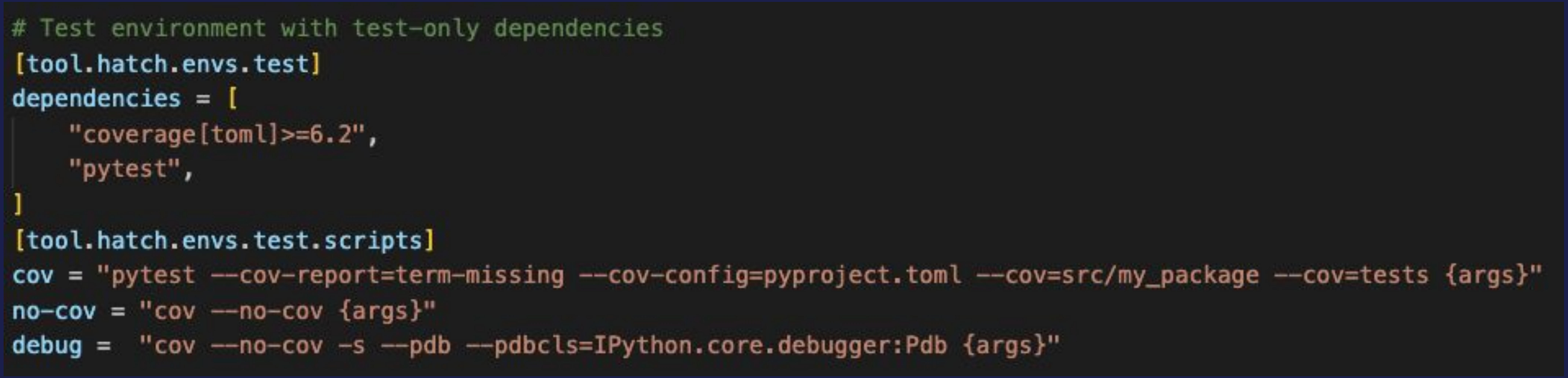
```toml
[build-system]
requires = ["hatchling", "hatch-vcs"]
build-backend = "hatchling.build"

[project]
name = "my-python-project"
description = "Streamlined Python Project"
readme = "README.md"
requires-python = ">=3.10"
license = "MIT"
keywords = ["keyword_1", "keyword_2"]  # for PyPI
authors = [
    { name = "Florian Wilhelm", email = "email@example.com" },
]
classifiers = [ # options under https://pypi.org/classifiers/
    "Development Status :: 2 - Pre-Alpha",
    "Programming Language :: Python",
]
dependencies = [ # direct dependencies of this package
    "typer",
    "numpy",
]
version = "1.0"

[tool.hatch.build]
packages = ["src/my_package"]
```

inovex

Automation with Scripts!

Scripts in pyproject.toml for automation of tasks, e.g.

- running unit-tests with our without coverage, debugging,
- building the documentation,
- running the linters, code checks, mypy,
- …



```toml
# Test environment with test-only dependencies
[tool.hatch.envs.test]
dependencies = [
    "coverage[toml]>=6.2",
    "pytest",
]
[tool.hatch.envs.test.scripts]
cov = "pytest --cov-report=term-missing --cov-config=pyproject.toml --cov=src/my_package --cov=tests {args}"
no-cov = "cov --no-cov {args}"
debug =  "cov --no-cov -s --pdb --pdbcls=IPython.core.debugger:Pdb {args}"
```

```
> hatch run test:cov
```

inovex

Code Quality: Linting & Formatting

Black

isort

flake8

autoflake

pydocstyle

...

Ruff

- replaces tons of tools
- easy configuration via pyproject.toml
- extremely fast
- over 700 plugins

inovex

my[py]

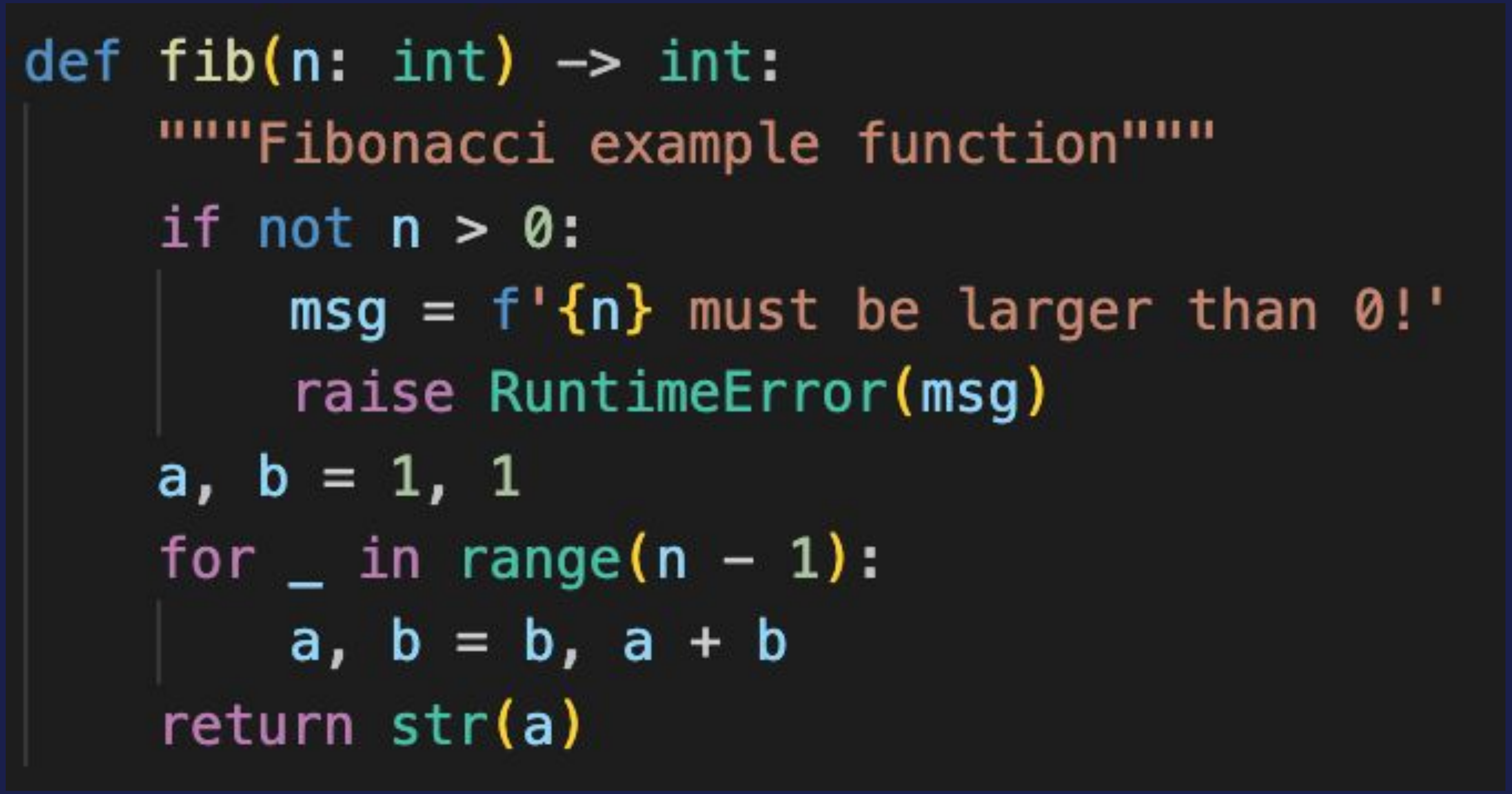
Why mypy?

compile-time type checking finds many errors in advance, often edge cases.

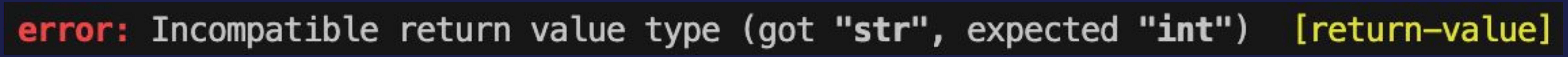
type declaration act as machine-checked documentation, thus enhancing the dev experience.

inovex

Mypy Example


```python
def fib(n: int) -> int:
    """Fibonacci example function"""
    if not n > 0:
        msg = f'{n} must be larger than 0!'
        raise RuntimeError(msg)
    a, b = 1, 1
    for _ in range(n - 1):
        a, b = b, a + b
    return str(a)
```

```
> hatch run lint:typing
```


```
error: Incompatible return value type (got "str", expected "int")  [return-value]
```

inovex

Testing with pytest & hatch

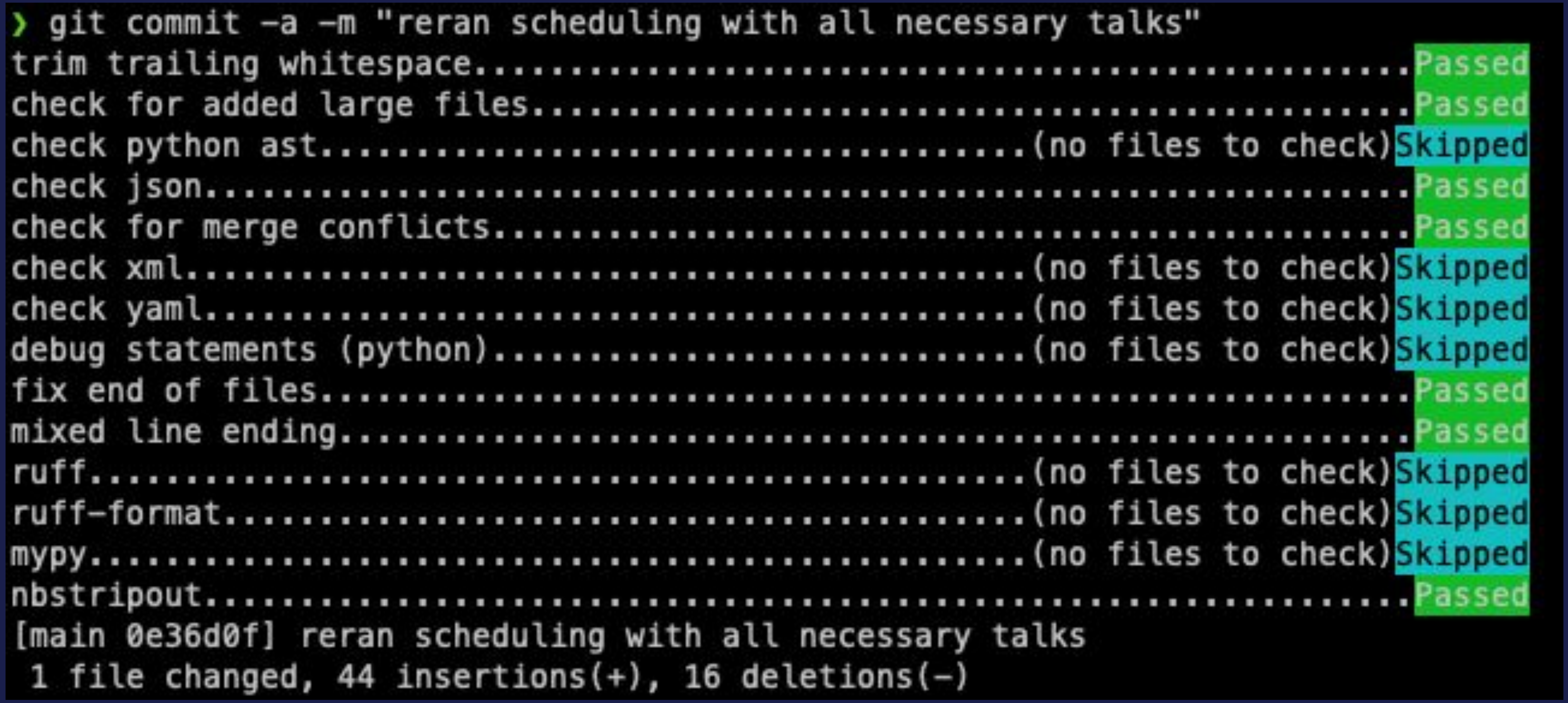
pytest

- defacto standard for unit testing
- powerful features like fixtures, etc.
- tons of useful plugins, e.g.:
  - pytest-cov for coverage
  - pytest-recording for mocking calls to external services
  - pytest-sugar to make it easier on the eyes

hatch & tox

- isolated environments for testing different Python versions and dependency combinations

Avoiding human-errors by automated checks on every git commit



```
❯ git commit -a -m "reran scheduling with all necessary talks"
trim trailing whitespace.................................................Passed
check for added large files..............................................Passed
check python ast.................................(no files to check)Skipped
check json...............................................................Passed
check for merge conflicts................................................Passed
check xml........................................(no files to check)Skipped
check yaml.......................................(no files to check)Skipped
debug statements (python)........................(no files to check)Skipped
fix end of files.........................................................Passed
mixed line ending........................................................Passed
ruff.............................................(no files to check)Skipped
ruff-format......................................(no files to check)Skipped
mypy.............................................(no files to check)Skipped
nbstripout...............................................................Passed
[main 0e36d0f] reran scheduling with all necessary talks
 1 file changed, 44 insertions(+), 16 deletions(-)
```

inovex

Automation with CI/CD

- Automatic and reproducible testing
- Publishing packages based on git tags
- Established branching strategy, e.g. GithubFlow for efficient collaboration
- Scalability and Adaptability when needed
- Automated deployments, building of documentation etc.

GitLab

GitHub

Conclusion

- unified configuration in **pyproject.toml**
- standardized folder structure with **src-layout** and useful **README.md**
- easy package management and automation with **hatch**
- automated QA with **ruff**, **pytest**, **pre-commit**, **mypy**, CI/CD
- proper documentation with **mkdocs**
- **automation & conventions** are key!

inovex

Check out the Hatchlor!

The Hatchlor

⭐https://github.com/FlorianWilhelm/the-hatchlor

inovex

CHEERS TO THE COMMUNITY

Credits & Resources

- Ofek Lev, the creator of hatch, for is awesome work in his spare time ❤️
- Michael Hofmann from inovex who made these awesome slides

inovex

PyConDE / PyData 2024

Thank you!

Dr. Florian Wilhelm

Head of Data Science

florian.wilhelm@inovex.de

inovex.de

@inovexlife

@inovexgmbh

inovex