Professionelle Python-Projekte – Packaging, Tooling und Distribution

Python-Projekte strukturieren, paketieren und zuverlässig verteilen

01 INTRO

Zwischen einem funktionierenden Python-Programm und einem professionell nutzbaren Softwarepaket liegen einige zusätzliche Schritte. Abhängigkeiten müssen verwaltet, Projekte reproduzierbar gebaut, Versionen vergeben und Software zuverlässig auf Entwicklungs-, Test- und Produktivsysteme verteilt werden können.

Der Kurs zeigt, wie Python-Projekte mit aktuellen Werkzeugen strukturiert, gebaut und verteilt werden. Im Mittelpunkt stehen moderne Projektkonfiguration mit pyproject.toml, Packaging, Dependency Management, automatisierte Qualitätsprüfungen und ein nachvollziehbarer Build- und Releaseprozess.

DAUER

3 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Sichere Python-Kenntnisse und praktische Erfahrung mit Python-Projekten

02 TRAININGSPROGRAMM

Inhalte

Vom Skript zum Python-Projekt

- Skript, Modul, Package und Distribution
- typische Projektstrukturen
- Source Layout und Flat Layout
- öffentliche und interne Module
- Paketgrenzen
- Imports
- ausführbare Anwendungen und Bibliotheken
- Projektstruktur bewusst wählen

Moderne Projektkonfiguration

- pyproject.toml
- Projektmetadaten
- Dependencies
- optionale Dependencies
- Development Dependencies
- Tool-Konfiguration
- zentrale statt verteilter Konfiguration
- bestehende Projekte modernisieren

Virtuelle Umgebungen

- Zweck virtueller Umgebungen
- venv
- unterschiedliche Python-Versionen
- reproduzierbare Entwicklungsumgebungen
- Projekt und Umgebung trennen
- Entwicklungs-, Test- und Produktionsumgebungen
- typische Probleme mit Python-Umgebungen

Dependency Management

- direkte und transitive Abhängigkeiten
- Versionsangaben
- kompatible Versionsbereiche
- Lockfiles
- reproduzierbare Installationen
- Dependency-Konflikte
- Runtime- und Development-Dependencies
- Abhängigkeiten bewusst begrenzen

Python-Werkzeuge einordnen

- pip
- pipx
- uv
- unterschiedliche Packaging- und Dependency-Tools
- Toolchains vergleichen
- Werkzeugauswahl nach Projektanforderung
- vorhandene Toolchains nicht unnötig ersetzen
- Abhängigkeit vom Werkzeug vermeiden

Packages bauen

- Build-Systeme in Python
- Build Backends
- Source Distributions
- Wheels
- Paket erzeugen
- Build-Artefakte untersuchen
- Dateien in Distributionen aufnehmen
- Paket lokal installieren und prüfen

Ressourcen und Paketdaten

- Konfigurationsdateien
- Templates und andere Ressourcen
- Package Data
- importlib.resources
- Pfade innerhalb installierter Packages
- Unterschiede zwischen Source Tree und Installation
- typische Fehler beim Zugriff auf Ressourcen

Kommandozeilenprogramme verteilen

- Python-Anwendungen als CLI
- Entry Points
- Console Scripts
- Installation ausführbarer Befehle
- Argumentverarbeitung
- Versionsinformationen
- pipx für Kommandozeilenwerkzeuge
- interne Tools verteilen

Versionierung

- Versionsnummern
- Semantic Versioning
- Release- und Development-Versionen
- Pre-Releases
- Versionsinformationen im Package
- Git Tags
- Version und Build miteinander verbinden
- nachvollziehbare Releases

Private und öffentliche Package-Repositories

- PyPI
- TestPyPI
- private Package-Repositories
- interne Unternehmenspakete
- Paketquellen konfigurieren
- Authentifizierung
- Pakete veröffentlichen
- vorhandene Versionen und Artefakte verwalten

Qualität automatisieren

- Formatter
- Linter
- Type Checker
- automatisierte Tests
- zentrale Tool-Konfiguration
- Qualitätsprüfungen reproduzierbar ausführen
- lokale und zentrale Prüfungen angleichen
- Quality Gates mit Augenmaß

Pre-Commit Checks

- pre-commit
- Checks vor dem Commit
- Formatierung
- Linting
- einfache Qualitätsprüfungen
- Hooks konfigurieren
- lokale Entwicklungsabläufe vereinheitlichen
- Grenzen automatischer Checks

Build und Test in CI

- reproduzierbare Builds
- saubere Build-Umgebungen
- Tests automatisiert ausführen
- mehrere Python-Versionen
- Build-Artefakte erzeugen
- Paket vor Veröffentlichung testen
- Artefakte zwischen Pipeline-Stufen
- grundlegende CI/CD-Struktur

Releaseprozess

- Änderungen für ein Release vorbereiten
- Version festlegen
- Tests und Qualitätsprüfungen
- Paket bauen
- Artefakte prüfen
- veröffentlichen
- Git Tags und Releaseinformationen
- automatisierte Releases
- Fehler im Releaseprozess behandeln

Interne Python-Software verteilen

- Bibliothek oder Anwendung?
- zentrale interne Packages
- gemeinsame Unternehmensbibliotheken
- kleine Kommandozeilenwerkzeuge
- private Repositories
- Versionsabhängigkeiten zwischen Projekten
- Updates kontrolliert verteilen
- Wildwuchs aus kopierten Skripten vermeiden

Dokumentation und Metadaten

- README
- Installationsanleitung
- Nutzungsbeispiele
- Package-Metadaten
- Projekt-URLs
- Python-Versionen und Abhängigkeiten dokumentieren
- Changelog
- Dokumentation als Teil des Releases

Bestehende Projekte modernisieren

- gewachsene Projektstrukturen analysieren
- Abhängigkeiten erfassen
- requirements.txt und bestehende Konfigurationen
- schrittweise zu pyproject.toml
- Packaging nachträglich einführen
- Tests und Qualitätswerkzeuge integrieren
- Migration ohne unnötigen Big Bang

03 DURCHFÜHRUNG

Rahmen und Organisation

Dauer	3 Tage
Durchführung	Präsenz oder Live Online, vorzugsweise als Inhouse-Schulung
Schwerpunkt	Inhouse-Schulungen für Unternehmen und Teams
Sprache	Deutsch, englische Fachbegriffe und Dokumentation
Unterlagen	Kursunterlagen, Beispiele und Übungsaufgaben
Technik	Eigener Rechner mit aktueller Python-Version und Entwicklungsumgebung

> **Inhouse-Schulungen**
> Die Schwerpunkte können an vorhandene Kenntnisse, eingesetzte Technologien und konkrete Anforderungen des Teams angepasst werden.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Der Kurs richtet sich an Python-Entwicklerinnen und -Entwickler, die Python-Software nicht nur entwickeln, sondern strukturiert bauen, versionieren und innerhalb eines Teams oder Unternehmens verteilen möchten.

Er eignet sich insbesondere für Teams mit mehreren Python-Projekten, internen Bibliotheken oder gewachsenen Sammlungen von Skripten, die ihre Entwicklungs-, Build- und Releaseprozesse vereinheitlichen möchten.

Die Teilnehmer sollten über praktische Python-Erfahrung verfügen und mit folgenden Themen sicher umgehen können:

- Module und Packages
- virtuelle Umgebungen
- Installation von Python-Paketen
- grundlegender Umgang mit Git
- automatisierte Tests in Grundzügen

> **Packaging-Erfahrung**
> Erfahrungen mit Packaging oder CI/CD werden nicht vorausgesetzt.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- Python-Projekte sinnvoll strukturieren
- moderne Projekte mit pyproject.toml konfigurieren
- Abhängigkeiten und Entwicklungsumgebungen reproduzierbar verwalten
- geeignete Packaging- und Dependency-Werkzeuge auswählen
- Wheels und Source Distributions erzeugen
- Ressourcen korrekt in Packages integrieren
- Python-Bibliotheken und Kommandozeilenprogramme paketieren
- Versions- und Releasekonzepte anwenden
- Pakete über öffentliche oder private Repositories verteilen
- Qualitätsprüfungen in den Entwicklungsworkflow integrieren
- Builds und Tests automatisieren
- nachvollziehbare Releaseprozesse aufbauen
- bestehende Python-Projekte schrittweise modernisieren

06 FAQ

Häufige Fragen

Ist der Kurs für Python-Einsteiger geeignet?

Nein. Der Kurs setzt praktische Python-Erfahrung voraus. Er beschäftigt sich nicht mit der Sprache selbst, sondern mit der Organisation und Distribution professioneller Python-Projekte.

Wird pyproject.toml ausführlich behandelt?

Ja. pyproject.toml bildet den zentralen Ausgangspunkt für moderne Python-Projekte und wird sowohl für Projektmetadaten als auch für Build-, Dependency- und Tool-Konfiguration verwendet.

Welches Packaging-Werkzeug wird verwendet?

Der Kurs vermittelt zunächst die zugrunde liegenden Python-Standards und ordnet darauf aufbauende Werkzeuge ein. Die konkrete Toolchain kann an die im Unternehmen eingesetzten Werkzeuge angepasst werden.

Wird uv behandelt?

Ja. uv wird als aktuelle Toolchain eingeordnet und praktisch eingesetzt. Gleichzeitig wird darauf geachtet, die zugrunde liegenden Python-Mechanismen zu verstehen und den Kurs nicht von einem einzelnen Werkzeug abhängig zu machen.

Wird PyPI verwendet?

Der Veröffentlichungsprozess kann mit TestPyPI demonstriert werden. Ebenso werden private Package-Repositories und die Verteilung interner Unternehmenssoftware behandelt.

Geht es auch um CI/CD?

Ja, soweit Build, Test und Veröffentlichung eines Python-Pakets betroffen sind. Allgemeiner Aufbau und Betrieb komplexer CI/CD-Plattformen sind nicht Schwerpunkt des Kurses.

Ist der Kurs für Unternehmen mit vielen internen Python-Skripten geeignet?

Gerade dafür kann er sinnvoll sein. Ein Schwerpunkt kann darauf gelegt werden, aus einzeln verteilten Skripten versionierte, installierbare und zentral verwaltete Python-Werkzeuge und Bibliotheken zu entwickeln.

Kann unsere bestehende Python-Toolchain berücksichtigt werden?

Ja. Bei Inhouse-Schulungen können vorhandene Projektstrukturen, Package-Repositories, CI/CD-Systeme und eingesetzte Werkzeuge in die Schulung einbezogen werden.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/python-packaging/