Python für Fortgeschrittene

Objektorientierung, Tools und Best Practice

Programmierkenntnisse erweisen sich, unabhängig vom Anwendungsfall, als zunehmend wertvolle Fähigkeit. Python ist hierbei eine der aktuell am weitest verbreiteten und gefragten Programmiersprachen. Wenn Sie bereits mit den ersten Grundlagen von Python vertraut sind, bringt Sie dieses Python Training auf den nächsten Level.

Sie werden zu Beginn Grundelemente der objektorientierten Programmierung, wichtige Bausteine der Standardbibliothek und tiefergehende Softwarekonzepte kennen lernen. Software unterliegt immer strengeren Qualitätsstandards. Daher werden wir uns mit Themen wie Python Code Design und Code Analysetools beschäftigen.

Python kann auf die unterschiedlichsten Datenquellen zugreifen und Prozesse automatisieren. Am Beispiel von Rest API Abfragen und Datenbankanbindungen werden wir Abfragen und Berechnung durchführen, ohne dabei das Rad neu zu erfinden. Wir bedienen uns vorgefertigter Module und passen diese an unsere Bedürfnisse an.

Kursinhalt

- Grundkonzepte der Objektorientierung
- Dekoratoren, Generatoren
- Wichtige Elemente der Standardbibliotheken
- Fehlerbehebung, Debugging, Logfiles
- Paket- und Abhängigkeitsverwaltung
- Virtuelle Python-Umgebungen
- Statische Codeanalyse
- Testautomatisierung, Testbibliotheken
- GIT Integration in Visual Studio Code

E-Book Das ausführliche deutschsprachige digitale Unterlagenpaket, bestehend aus PDF und E-Book, ist im Kurspreis enthalten.

Zielgruppe

Der Kurs richtet sich an alle, die bereits erste Programmierkenntnisse besitzen und sich nun weiter mit Python und der objektorientierten Programmierung beschäftigen wollen und die vor ersten Projektaufgaben stehen, die sie mit Python realisieren möchten.

Voraussetzungen

Sie benötigen für diesen Kurs bereits solide Python-Grundlagen in Bezug auf Datentypen, Funktionen und Schleifenkonstruktionen. Diese können z. B. in unserem Kurs Python für Einsteiger – Einführung in die Programmierung erworben werden.

Dieser Kurs im Web

Alle tagesaktuellen Informationen und Möglichkeiten zur Bestellung finden Sie unter dem folgenden Link: www.experteach.ch/go/**PYFI**

Vormerkung

Sie können auf unserer Website einen Platz kostenlos und unverbindlich für 7 Tage reservieren. Dies geht auch telefonisch unter 06074 4868-0.

Garantierte Kurstermine

Für Ihre Planungssicherheit bieten wir stets eine große Auswahl garantierter Kurstermine an.

Ihr Kurs maßgeschneidert

Diesen Kurs können wir für Ihr Projekt exakt an Ihre Anforderungen anpassen.

Training	Preise zzgl. MwSt.
Termine in Deutschland	5 TageCHF 3.295,-
Termine in Österreich	5 TageCHF 3.295,-
Online Training	5 TageCHF 3.295,-

Termin/Kursort — Kurssprache Deutsch

25.08.-29.08.25	Frankfurt	26.01.-30.01.26	Online
25.08.-29.08.25	Online	23.03.-27.03.26	Düsseldorf
06.10.-10.10.25	Düsseldorf	23.03.-27.03.26	Online
06.10.-10.10.25	Online	04.05.-08.05.26	Online
17.11.-21.11.25	Online	04.05.-08.05.26	Wien
17.11.-21.11.25	Wien	22.06.-26.06.26	München
26.01.-30.01.26	Frankfurt	22.06.-26.06.26	Online

Stand 24.06.2025

Python für Fortgeschrittene

Inhaltsverzeichnis

Python für Fortgeschrittene – Objektorientierung, Tools und Best Practice

1 Wiederholung: Special Concepts
1.1 Annotations, Typehints und Type-Checking
1.1.1 Typehints in Funktionen
1.1.2 Konventionen und Funktionen von Typehints
1.1.3 JSON korrekt typisieren mit TypedDict
1.1.4 Type-Checking mit MyPy
1.1.5 Type-Checking in VSC
1.2 Lambda Funktionen
1.3 Funktionale Konzepte
1.3.1 map() und filter()
1.3.2 Das Modul functools
1.3.3 Das Modul operator
1.4 Comprehensions
1.5 Format-Strings

2 Logging
2.1 Grundlagen Logging
2.1.1 Einfache Basis-Konfiguration
2.1.2 Verfügbare Logging-Level
2.1.3 Ausgaben formatieren
2.2 Fortgeschrittenes Logging
2.2.1 Loggers
2.2.2 Handlers
2.2.3 Formatters
2.2.4 Filters
2.2.5 Logger konfigurieren
2.2.6 Exceptions loggen

3 Objektorientierte Programmierung
3.1 Die Probleme der prozeduralen Programmierung
3.1.1 Ein prozedurales Antibeispiel
3.2 Einführung in die OOP
3.2.1 OOP vs Prozedurale Programmierung
3.3 Klassen und Objekte
3.3.1 Definition und Deklaration von Klassen
3.3.2 Attribute und Methoden von Objekten
3.4 Vererbung
3.4.1 Vererbungs-Hierarchien
3.4.2 Methodenüberschreibung
3.4.3 Zugriff auf die Superklasse
3.5 Polymorphie
3.5.1 Operatorüberladung
3.6 Datenkapselung
3.6.1 Zugriffsmodifikatoren
3.6.2 Getter und Setter Methoden
3.6.3 Property Dekoratoren in Python
3.7 Statische Methoden
3.8 Klassenmethoden

4 Dekoratoren
4.1 Einführung
4.2 Benötigte Konzepte
4.2.1 Funktions-Referenzen
4.2.2 Funktionen in Funktionen
4.2.3 Funktionen als Übergabeparameter
4.2.4 Funktionen als Rückgabewert
4.3 Einfache Dekoratoren
4.3.1 Die Wrapper-Funktion
4.3.2 Eine Funktion dekorieren
4.4 Anwendungsfälle von Dekoratoren
4.4.1 Argumentüberprüfung
4.4.2 Funktionsaufrufe zählen
4.4.3 Aufruf einer Funktion zeigen (Logging)
4.5 Dekoratoren mit Übergabeparametern
4.6 Klassen als Dekoratoren
4.6.1 Eine Klasse als Dekorator benutzen
4.6.2 Dekorator-Klassen mit Übergabeparametern
4.7 Typische Dekoratoren und das Modul functools
4.7.1 Functools Dekoratoren

5 Generatoren und Iteratoren
5.1 yield und next()
5.1.1 Verwendung von next()
5.1.2 Beispiele
5.2 return vs yield
5.3 Subgeneratoren mit yield from
5.4 Generator-Expressions
5.5 Speicheroptimierung vs Laufzeit-Optimierung
5.6 Kommunikation mit Generatoren
5.6.1 Exceptions als Sentinels
5.6.2 Details zum Generator-Typehint
5.7 Klassenbasierter Ansatz: Iteratoren
5.8 Generatoren der Standard Library
5.8.1 Filternde Generatoren
5.8.2 Abbildende Generatoren (Mapping)
5.8.3 Zusammenfügende Generatoren (Mergers)
5.8.4 Expandierende Generatoren
5.8.5 Umordnende Generatoren
5.9 Ausblick: Asynchrone Programmierung
5.9.1 Erster Ansatz: How to Async
5.9.2 Zweiter Ansatz: Tasks zu echter Asynchronität
5.9.3 Dritter Ansatz: Skalierbarkeit über Tasklisten
5.9.4 Generator oder Coroutine?

6 Unittest, Doctest
6.1 Unittest
6.1.1 unittest - Die Klasse TestCase
6.1.2 assert Methoden
6.1.3 setUp und tearDown Methoden
6.1.4 Testen von Fehlermeldungen
6.1.5 Überspringen und erwartete Fehlschläge
6.2 doctest - Docstrings für Tests nutzen
6.2.1 Einfache Verwendung von doctest
6.2.2 Die Ausgabe von doctest
6.2.3 Testen der Methoden einer Klasse
6.2.4 Tests in einer Textdatei
6.3 Test Driven Development (TDD) – Tests zuerst
6.4 Static Code Analysis
6.4.1 Linting mit pylint
6.4.2 Type Checks mit mypy

7 Ausnahmebehandlung
7.1 Exceptions in Python
7.2 Hierarchie der Builtin Exceptions (Ausschnitt)
7.3 Eigene Exceptions definieren
7.4 Exception Chaining

8 Skriptaufrufe
8.1 Die Kommandozeilen-Optionen
8.1.1 Module aufrufen python -m
8.1.2 pdb: Der Python Debugger
8.1.3 Das timeit-Modul zur Zeitmessung
8.1.4 Mit JSON-Dateien arbeiten: json.tool
8.1.5 Weitere Beispiele: compileall und tkinter
8.2 Argumente
8.2.1 Die Liste sys.argv
8.2.2 Argumente parsen mit dem argparse-Modul
8.3 Die Shebang Line
8.4 Module und Pakete
8.4.1 Top-level code environment
8.4.2 Verwendung der Variable __name__
8.4.3 Wichtige Dateien in Paketen

9 GUI mit TKinter und PyQt5
9.1 GUI mit Tkinter
9.2 Die Alles-In-Einem-File-Variante
9.3 PyGubu – Ein Wysiwyg - Editor für Tkinter
9.4 GUI mit PyQt5
9.4.1 PyQt5, PySide2 und Lizenzierung
9.4.2 Erster Start von PyQt5-Designer
9.4.3 Die Oberfläche von Qt-Designer
9.4.4 Layouts in Qt Designer
9.4.5 PyQt5-Designs in Python-Code laden
9.4.6 Buttons und Methoden verknüpfen

10 Virtuelle Umgebungen
10.1 Abhängigkeiten von Modulen
10.2 Das Konzept einer virtuellen Umgebung
10.3 Das Tool venv

11 Eigene Packages und der PyPI
11.1 Angeleitete Übung zur Einführung von Packages
11.1.1 Relative Importe - Schritt 2
11.1.2 Eigenes Paket mit __init__.py - Schritt 3
11.2 Eigene Packages auf PyPI veröffentlichen
11.2.1 setup.py
11.2.2 Erstellung der Distribution
11.2.3 Auf PyPI veröffentlichen
11.2.4 Wichtige Zusatzinformationen

12 Code Distribution
12.1 Distribution mit Python Setuptools
12.2 Das Distributions Problem
12.3 Distribution mit PyInstaller
12.3.1 Vorbereitung und Build-Erzeugnisse
12.3.2 Funktionsweise und One-File
12.3.3 Kommandozeilenargumente für PyInstaller
12.3.4 PyInstaller mit Python-Code ausführen

A Visual Studio Code und Jupyter
A.1 Was ist Visual Studio Code?
A.1.1 Visual Studio Code vs. Visual Studio
A.1.2 Visual Studio Code unter Windows installieren
A.1.3 Optionen während der Installation
A.1.4 Das Python-Extension Pack für VS Code
A.1.5 Sprachunterstützung von VS Code
A.1.6 Visual Studio Code updaten
A.1.7 Die Oberfläche von VS Code
A.1.8 Die Oberfläche von VS Code - Activity Bar und Sidebar
A.1.9 Das Search Tool
A.1.10 Die Git-Leiste
A.1.11 Git in Visual Studio Code nutzen
A.1.12 Ein Remoterepository in VS Code klonen
A.1.13 Ein lokales Git-Repository anlegen
A.1.14 Lokales Repository mit Remoterepository synchronisieren
A.1.15 GitLens
A.1.16 Neue Dateien in VS Code anlegen
A.1.17 Mit einzelnen Codedateien in Visual Studio Code arbeiten
A.1.18 Visual Studio Code einen Ordner hinzufügen
A.1.19 Mit Workspaces in VS Code arbeiten
A.1.20 Struktur von Workspaces
A.1.21 Auf Remote Terminals entwickeln
A.1.22 Debugging in Visual Studio Code
A.1.23 Einen Linter für Python in VS Code nutzen
A.1.24 Linter in Aktion
A.1.25 Python Code in Visual Studio Code testen
A.1.26 Ein Testframework in VSCode aktivieren
A.1.27 Tests erzeugen und durchführen
A.2 Jupyter Notebooks
A.2.1 Interaktive Code-Zellen
A.2.2 Grafische Oberflächen funktionieren
A.2.3 Exporte in andere Formate und Hilfen
A.2.4 Editieren und Ausführen von Zellen
A.2.5 Shortcuts sind im Markdown Modus gefährlich
A.2.6 Dynamische HTML Seiten einbetten
A.2.7 Variable Inspector als Debugger
A.3 Anaconda - Scientific Power Package für ML
A.3.1 Die Anaconda Foren/Learning - Dashboards
A.4 Jupyter Notebooks im VSC

ExperTeach AG
Kronenstrasse 11 · 8735 St. Gallenkappel · Telefon: +41 55 420 2591 · Fax: +41 55 420 2592 · info@experteach.ch · www.experteach.ch