

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 tiefere 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 Datenbankverbindungen 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 Tage CHF 3.295,-
Termine in Österreich	5 Tage CHF 3.295,-
Online Training	5 Tage CHF 3.295,-
Termin/Kursort	Kurssprache Deutsch
19.05.-23.05.25 Online	06.10.-10.10.25 Düsseldorf
07.07.-11.07.25 Düsseldorf	06.10.-10.10.25 Online
07.07.-11.07.25 Online	17.11.-21.11.25 Online
25.08.-29.08.25 Frankfurt	17.11.-21.11.25 Wien
25.08.-29.08.25 Online	

Stand 07.05.2025



Inhaltsverzeichnis

Python für Fortgeschrittene – Objektorientierung, Tools und Best Practice

1 Wiederholung: Special Concepts	5 Generatoren und Iteratoren	9.4.3 Die Oberfläche von Qt-Designer
1.1 Annotations, Typehints und Type-Checking	5.1 yield und next()	9.4.4 Layouts in Qt Designer
1.1.1 Typehints in Funktionen	5.1.1 Verwendung von next()	9.4.5 PyQt5-Designs in Python-Code laden
1.1.2 Konventionen und Funktionen von Typehints	5.1.2 Beispiele	9.4.6 Buttons und Methoden verknüpfen
1.1.3 JSON korrekt typisieren mit TypedDict	5.2 return vs yield	10 Virtuelle Umgebungen
1.1.4 Type-Checking mit MyPy	5.3 Subgeneratoren mit yield from	10.1 Abhängigkeiten von Modulen
1.1.5 Type-Checking in VSC	5.4 Generator-Expressions	10.2 Das Konzept einer virtuellen Umgebung
1.2 Lambda Funktionen	5.5 Speicheroptimierung vs Laufzeit-Optimierung	10.3 Das Tool venv
1.3 Funktionale Konzepte	5.6 Kommunikation mit Generatoren	11 Eigene Packages und der PyPI
1.3.1 map() und filter()	5.6.1 Exceptions als Sentinels	11.1 Angeleitete Übung zur Einführung von Packages
1.3.2 Das Modul functools	5.6.2 Details zum Generator-Typehint	11.1.1 Relative Importe - Schritt 2
1.3.3 Das Modul operator	5.7 Klassenbasierter Ansatz: Iteratoren	11.1.2 Eigenes Paket mit <code>__init__.py</code> - Schritt 3
1.4 Comprehensions	5.8 Generatoren der Standard Library	11.2 Eigene Packages auf PyPI veröffentlichen
1.5 Format-Strings	5.8.1 Filternde Generatoren	11.2.1 setup.py
2 Logging	5.8.2 Abbildende Generatoren (Mapping)	11.2.2 Erstellung der Distribution
2.1 Grundlagen Logging	5.8.3 Zusammenfügende Generatoren (Mergers)	11.2.3 Auf PyPI veröffentlichen
2.1.1 Einfache Basis-Konfiguration	5.8.4 Expandierende Generatoren	11.2.4 Wichtige Zusatzinformationen
2.1.2 Verfügbare Logging-Level	5.8.5 Umordnende Generatoren	12 Code Distribution
2.1.3 Ausgaben formatieren	5.9 Ausblick: Asynchrone Programmierung	12.1 Distribution mit Python Setuptools
2.2 Fortgeschrittenes Logging	5.9.1 Erster Ansatz: How to Async	12.2 Das Distributions Problem
2.2.1 Loggers	5.9.2 Zweiter Ansatz: Tasks zu echter Asynchronität	12.3 Distribution mit Pynstaller
2.2.2 Handlers	5.9.3 Dritter Ansatz: Skalierbarkeit über Tasklisten	12.3.1 Vorbereitung und Build-Ergebnisse
2.2.3 Formatters	5.9.4 Generator oder Coroutine?	12.3.2 Funktionsweise und One-File
2.2.4 Filters	6 Unittest, Doctest	12.3.3 Kommandozeilenargumente für Pynstaller
2.2.5 Logger konfigurieren	6.1 Unittest	12.3.4 Pynstaller mit Python-Code ausführen
2.2.6 Exceptions loggen	6.1.1 unittest - Die Klasse TestCase	A Visual Studio Code und Jupyter
3 Objektorientierte Programmierung	6.1.2 assert Methoden	A.1 Was ist Visual Studio Code?
3.1 Die Probleme der prozeduralen Programmierung	6.1.3 setUp und tearDown Methoden	A.1.1 Visual Studio Code vs. Visual Studio
3.1.1 Ein prozedurales Antibeispiel	6.1.4 Testen von Fehlermeldungen	A.1.2 Visual Studio Code unter Windows installieren
3.2 Einführung in die OOP	6.1.5 Überspringen und erwartete Fehlschläge	A.1.3 Optionen während der Installation
3.2.1 OOP vs Prozedurale Programmierung	6.2 doctest - Docstrings für Tests nutzen	A.1.4 Das Python-Extension Pack für VS Code
3.3 Klassen und Objekte	6.2.1 Einfache Verwendung von doctest	A.1.5 Sprachunterstützung von VS Code
3.3.1 Definition und Deklaration von Klassen	6.2.2 Die Ausgabe von doctest	A.1.6 Visual Studio Code updaten
3.3.2 Attribute und Methoden von Objekten	6.2.3 Testen der Methoden einer Klasse	A.1.7 Die Oberfläche von VS Code
3.4 Vererbung	6.2.4 Tests in einer Textdatei	A.1.8 Die Oberfläche von VS Code - Activity Bar und Sidebar
3.4.1 Vererbungs-Hierarchien	6.3 Test Driven Development (TDD) – Tests zuerst	A.1.9 Das Search Tool
3.4.2 Methodenüberschreibung	6.4 Static Code Analysis	A.1.10 Die Git-Leiste
3.4.3 Zugriff auf die Superklasse	6.4.1 Linting mit pylint	A.1.11 Git in Visual Studio Code nutzen
3.5 Polymorphie	6.4.2 Type Checks mit mypy	A.1.12 Ein Remoterepository in VS Code klonen
3.5.1 Operatorüberladung	7 Ausnahmebehandlung	A.1.13 Ein lokales Git-Repository anlegen
3.6 Datenkapselung	7.1 Exceptions in Python	A.1.14 Lokales Repository mit Remoterepository synchronisieren
3.6.1 Zugriffsmodifikatoren	7.2 Hierarchie der Builtin Exceptions (Ausschnitt)	A.1.15 GitLens
3.6.2 Getter und Setter Methoden	7.3 Eigene Exceptions definieren	A.1.16 Neue Dateien in VS Code anlegen
3.6.3 Property Dekoratoren in Python	7.4 Exception Chaining	A.1.17 Mit einzelnen Codedateien in Visual Studio Code arbeiten
3.7 Statische Methoden	8 Skriptaufrufe	A.1.18 Visual Studio Code einen Ordner hinzufügen
3.8 Klassenmethoden	8.1 Die Kommandozeilen-Optionen	A.1.19 Mit Workspaces in VS Code arbeiten
4 Dekoratoren	8.1.1 Module aufrufen python -m	A.1.20 Struktur von Workspaces
4.1 Einführung	8.1.2 pdb: Der Python Debugger	A.1.21 Auf Remote Terminals entwickeln
4.2 Benötigte Konzepte	8.1.3 Das timeit-Modul zur Zeitmessung	A.1.22 Debugging in Visual Studio Code
4.2.1 Funktions-Referenzen	8.1.4 Mit JSON-Dateien arbeiten: json.tool	A.1.23 Einen Linter für Python in VS Code nutzen
4.2.2 Funktionen in Funktionen	8.1.5 Weitere Beispiele: compileall und tkinter	A.1.24 Linter in Aktion
4.2.3 Funktionen als Übergabeparameter	8.2 Argumente	A.1.25 Python Code in Visual Studio Code testen
4.2.4 Funktionen als Rückgabewert	8.2.1 Die Liste sys.argv	A.1.26 Ein Testframework in VSCode aktivieren
4.3 Einfache Dekoratoren	8.2.2 Argumente parsen mit dem argparse-Modul	A.1.27 Tests erzeugen und durchführen
4.3.1 Die Wrapper-Funktion	8.3 Die Shebang Line	A.2 Jupyter Notebooks
4.3.2 Eine Funktion dekorieren	8.4 Module und Pakete	A.2.1 Interaktive Code-Zellen
4.4 Anwendungsfälle von Dekoratoren	8.4.1 Top-level code environment	A.2.2 Grafische Oberflächen funktionieren
4.4.1 Argumentüberprüfung	8.4.2 Verwendung der Variable <code>__name__</code>	A.2.3 Exporte in andere Formate und Hilfen
4.4.2 Funktionsaufrufe zählen	8.4.3 Wichtige Dateien in Paketen	A.2.4 Editieren und Ausführen von Zellen
4.4.3 Aufruf einer Funktion zeigen (Logging)	9 GUI mit Tkinter und PyQt5	A.2.5 Shortcuts sind im Markdown Modus gefährlich
4.5 Dekoratoren mit Übergabeparametern	9.1 GUI mit Tkinter	A.2.6 Dynamische HTML Seiten einbetten
4.6 Klassen als Dekoratoren	9.2 Die Alles-In-Einem-File-Variante	A.2.7 Variable Inspector als Debugger
4.6.1 Eine Klasse als Dekorator benutzen	9.3 PyGubu – Ein Wysiwyg - Editor für Tkinter	A.3 Anaconda - Scientific Power Package für ML
4.6.2 Dekorator-Klassen mit Übergabeparametern	9.4 GUI mit PyQt5	A.3.1 Die Anaconda Foren/Learning - Dashboards
4.7 Typische Dekoratoren und das Modul functools	9.4.1 PyQt5, PySide2 und Lizenzierung	A.4 Jupyter Notebooks im VSC
4.7.1 Functools Dekoratoren	9.4.2 Erster Start von PyQt5-Designer	

