

# 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** Sie erhalten das ausführliche deutschsprachige Unterlagenpaket aus der Reihe ExperTeach Networking – Print, E-Book und personalisiertes PDF! Bei Online-Teilnahme erhalten Sie das E-Book sowie das personalisierte PDF.

### 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.at/go/PYFI](http://www.experteach.at/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.	
<b>Termine in Deutschland</b>	<b>5 Tage</b>	<b>€ 2.995,-</b>
<b>Termine in Österreich</b>	<b>5 Tage</b>	<b>€ 2.995,-</b>
<b>Online Training</b>	<b>5 Tage</b>	<b>€ 2.995,-</b>
<b>Termin/Kursort</b>	Kurssprache <b>Deutsch</b>	
09.12.-13.12.24	Frankfurt	07.07.-11.07.25  Düsseldorf
09.12.-13.12.24	Online	07.07.-11.07.25  Online
03.02.-07.02.25	Frankfurt	25.08.-29.08.25  Frankfurt
03.02.-07.02.25	Online	25.08.-29.08.25  Online
24.03.-28.03.25	Hamburg	06.10.-10.10.25  Düsseldorf
24.03.-28.03.25	Online	06.10.-10.10.25  Online
19.05.-23.05.25	Online	17.11.-21.11.25  Online
19.05.-23.05.25	Wien	17.11.-21.11.25  Wien

Stand 02.11.2024



# Inhaltsverzeichnis

## Python für Fortgeschrittene – Objektorientierung, Tools und Best Practice

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

