

# Python Programming Professional

Prüfungscode IFW-PYTHON-300-01 · Fassung 1.0 · Stand 18.09.2026

Python für anspruchsvolle Projekte: Closures und Decorators, Objekt- und Datenmodell, Generatoren und Exceptions, Nebenläufigkeit mit asyncio sowie Typisierung.

## Die Prüfung im Überblick

|                                                   |                                                   |
|---------------------------------------------------|---------------------------------------------------|
| Bearbeitungszeit<br><b>2 Stunden</b>              | Fragen<br><b>70</b>                               |
| Bestehensgrenze<br><b>70 %</b>                    | Fragetypen<br><b>Einfach- und Mehrfachauswahl</b> |
| Prüfungssprache<br><b>Deutsch</b>                 | Zertifikat gültig<br><b>36 Monate</b>             |
| Durchführung<br><b>Online, zeitlich überwacht</b> | Preis<br><b>99,00 €</b>                           |

## Themenblöcke

| Block         | Thema                                       | Gewichtung   | Fragen    |
|---------------|---------------------------------------------|--------------|-----------|
| <b>1</b>      | Funktionen, Closures und Decorators         | 10 %         | 7         |
| <b>2</b>      | Objekt- und Datenmodell                     | 30 %         | 21        |
| <b>3</b>      | Generatoren, Context Manager und Exceptions | 20 %         | 14        |
| <b>4</b>      | Nebenläufigkeit und asyncio                 | 20 %         | 14        |
| <b>5</b>      | Typisierung                                 | 20 %         | 14        |
| <b>Gesamt</b> |                                             | <b>100 %</b> | <b>70</b> |

Die Gewichtung gibt an, welchen Anteil ein Block an den Prüfungsfragen hat. Die Fragen werden beim Start zufällig und anteilig je Block aus dem Fragenpool gezogen.

## Funktionen, Closures und Decorators

*Funktionen als Objekte: wann Namen gebunden werden und wie Decorators Verhalten ergänzen.*

- 1.1** Späte Bindung in Closures erkennen  
Lambdas in Schleifen und Comprehensions, Standardargumente als Abhilfe
- 1.2** Decorators mit Parametern schreiben  
drei Ebenen, `functools.wraps`
- 1.3** Ergebnisse zwischenspeichern  
`functools.lru_cache` und die Voraussetzung hashbarer Argumente

## Objekt- und Datenmodell

*Wie Python Attribute auflöst, Methoden vererbt und Klassen erzeugt.*

- 2.1** Die Method Resolution Order anwenden  
Mehrfachvererbung, kooperatives `super()`
- 2.2** Attributzugriffe steuern  
Deskriptoren mit `__get__`, `__set__` und `__set_name__`; `__getattr__` gegenüber `__getattribute__`
- 2.3** Gleichheit und Hashbarkeit festlegen  
`__eq__` und `__hash__`, Objekte als Schlüssel in dict und set
- 2.4** Speicher und Unveränderlichkeit steuern  
`__slots__`, `@dataclass(frozen=True)`, `field(default_factory=...)`
- 2.5** Klassenerzeugung anpassen  
`__init_subclass__`, Registrierung von Unterklassen

## Generatoren, Context Manager und Exceptions

*Kontrollfluss jenseits von Schleifen und Funktionsaufrufen.*

- 3.1** Generatoren delegieren  
`yield from` und der Rückgabewert eines Subgenerators
- 3.2** Mit Generatoren kommunizieren  
`send()`, `throw()`, `close()`
- 3.3** Context Manager schreiben  
`@contextmanager`, Ausnahmen innerhalb des `with`-Blocks
- 3.4** Exceptions verketteten  
`raise ... from ...`, `__cause__`, `__context__`, `from None`

## Nebenläufigkeit und asyncio

*Die passende Form der Nebenläufigkeit für I/O- und CPU-lastige Aufgaben wählen.*

- 4.1** Den Global Interpreter Lock einordnen  
Threads für I/O, Prozesse für CPU-lastige Aufgaben
- 4.2** Koroutinen nebenläufig ausführen  
`async/await`, `asyncio.gather()`, `asyncio.create_task()`
- 4.3** Den Event Loop nicht blockieren  
`time.sleep()` gegenüber `asyncio.sleep()`, `asyncio.to_thread()`

## Typisierung

*Type Hints so einsetzen, dass Werkzeuge Fehler früh finden.*

- 5.1** Type Hints richtig einordnen  
keine Prüfung zur Laufzeit durch CPython, statische Prüfung mit mypy oder pyright
- 5.2** Strukturelle Typisierung nutzen  
`typing.Protocol`, `@runtime_checkable`
- 5.3** Generische Typen schreiben  
`TypeVar`, `list[int]`, `Optional[str]` gegenüber optionalen Argumenten

*Institut für Webentwicklung · unabhängiges Lernangebot ohne Verbindung zu den genannten Technologieanbietern. Maßgeblich ist die Fassung auf der Zertifizierungsseite.*