Fortgeschrittene OOP-Konzepte in Python

Effektive Nutzung fortgeschrittener Python-Konzepte in der Praxis

Hussam Alafandi

5. Februar 2025

Agenda

Dunder-Methoden und Operatorüberladung

Komposition vs. Vererbung

Polymorphismus & Duck Typing

Entwurfsmuster (Design Patterns)

Zusammenfassung und Ausblick

Dunder-Methoden und Operatorüberladung

Theorie: Dunder-Methoden

- Dunder-Methoden sind spezielle Methoden, deren Name mit und endet mit `__`.
- Sie ermöglichen die Integration eigener Klassen in die Python-Syntax.
- Beispiele: `__init__` (Konstruktor), `__str__` (String-Darstellung), `__add__` (Operatorüberladung).

Hinweis: Detaillierte Beispiele und Übungen findest du im Notebook.

Kurzes Beispiel: Person-Klasse

```python
class Person:
    def __init__(self, name, alter):
        self.name = name
        self.alter = alter

    def __str__(self):
        return f"{self.name}, {self.alter} Jahre alt"

p = Person("Anna", 30)
print(p)
```

Komposition vs. Vererbung

Theorie: Vererbung & Komposition

- Vererbung: Erlaubt es, eine neue Klasse auf Basis einer bestehenden zu erstellen.
  Vorteil: Wiederverwendung von Code.
  Nachteil: Kann zu starker Kopplung führen.
- Komposition: Eine Klasse verwendet andere Klassen als Attribute.
  Vorteil: Flexibles, lose gekoppeltes Design.

Weitere Details und Beispiele im Notebook.

Kurzes Beispiel: Fahrzeug und Auto

Beispiel für Vererbung:

```
1 class Fahrzeug:
2     def starten(self):
3         print("Fahrzeug startet")
4
5 class Auto(Fahrzeug):
6     pass
7
8 a = Auto()
9 a.starten()
```

Kurzes Beispiel: Fahrzeug und Auto

Beispiel für Komposition:

```
class Motor:
    def starten(self):
        print("Motor startet")

class AutoMitMotor:
    def __init__(self):
        self.motor = Motor()
    def starten(self):
        self.motor.starten()
```

Polymorphismus & Duck Typing

Theorie: Polymorphismus und Duck Typing

- Polymorphismus: Unterschiedliche Objekte reagieren auf denselben Methodenaufruf.
- Duck Typing: Es zählt, ob ein Objekt die benötigte Methode besitzt nicht dessen Typ.

Details und praktische Beispiele gibt es im Notebook.

Kurzes Beispiel: Duck Typing

```
1  def starte_fahrzeug(fahrzeug):
2      fahrzeug.fahre()
3
4  class Auto:
5      def fahre(self):
6          print("Auto fährt")
7
8  class Fahrrad:
9      def fahre(self):
10         print("Fahrrad fährt")
11
12 starte_fahrzeug(Auto())
13 starte_fahrzeug(Fahrrad())
```

Entwurfsmuster (Design Patterns)

Theorie: Entwurfsmuster

- Entwurfsmuster sind wiederverwendbare Lösungen für häufige Design-Probleme.
- In Python helfen sie, den Code wartbar und erweiterbar zu gestalten.
- Wichtige Muster: Singleton, Factory, Observer, Strategy.

Ausführliche Erklärungen, Codebeispiele und Übungen im Notebook.

Kurzes Beispiel: Singleton

```
class Singleton:
    _instance = None
    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance

s1 = Singleton()
s2 = Singleton()
print(s1 is s2)  % Ausgabe: True
```

Zusammenfassung und Ausblick

Zusammenfassung

- Dunder-Methoden: Integrieren eigene Klassen in die Python-Syntax.
- Komposition vs. Vererbung: Komposition bietet flexible, lose gekoppelte Designs.
- Polymorphismus & Duck Typing: Erlauben generischen Code, der auf den Fähigkeiten der Objekte basiert.
- Entwurfsmuster: Bieten Lösungen für wiederkehrende Probleme.

Fragen & Diskussion

Vielen Dank für Ihre Aufmerksamkeit!