

Python 3 - Fortgeschrittene Konzepte und Anwendungen

Fortgeschrittenes OOP in Python

Hussam Alafandi

February 4, 2025

Einleitung

Ziele für heute:

- Grundlagen der objektorientierten Programmierung (OOP) festigen.
- Fortgeschrittene Themen wie Kapselung, abstrakte Klassen und Methoden erlernen.
- OOP anwenden, um reale Probleme zu modellieren.
- Debugging und Testen von OOP-Code üben.

Kapselung

Was ist Kapselung?

Definition:

- Kapselung bedeutet, Daten und Methoden, die auf diese Daten zugreifen, in einer Einheit (Klasse) zu bündeln.
- Zugriff auf Daten durch private und öffentliche Attribute steuern.

Schlüsselkonzepte:

- Öffentliche Attribute: Von außerhalb der Klasse zugänglich.
- Private Attribute: Von außerhalb der Klasse verborgen (Verwendung des Präfixes '__').

Vorteile der Kapselung:

- Schutz der Datenintegrität.
- Erleichterung von Änderungen und Erweiterungen.

Beispiel für Kapselung

Codebeispiel:

```
1 class BankAccount:
2     def __init__(self, owner, balance):
3         self.owner = owner
4         self.__balance = balance # Privates Attribut
5
6     def deposit(self, amount):
7         if amount > 0:
8             self.__balance += amount
9
10    def get_balance(self):
11        return self.__balance
12
13 account = BankAccount("Alice", 1000)
14 print(account.get_balance()) # Zugriff auf private Daten über Methode
```

Diskussion:

- Warum ist der direkte Zugriff auf `'__balance'` nicht erlaubt?
- Wie kann dies zu sicherem Programmieren beitragen?

Diskussion:

- Warum ist der direkte Zugriff auf `'__balance'` nicht erlaubt?
 - In Python wird `'__balance'` durch Namensmangling in `'_ClassName__balance'` umbenannt.
 - Dadurch wird verhindert, dass der Wert direkt von außerhalb der Klasse geändert wird.
- Wie kann dies zu sicherem Programmieren beitragen?
 - Verhindert unkontrollierte Manipulation von sensiblen Daten.
 - Erzwingt den Zugriff über getter- und setter-Methoden, die zusätzliche Prüfungen ermöglichen.

Abstrakte Klassen

Was sind abstrakte Klassen?

Definition:

- Abstrakte Klassen dienen als Blaupausen für andere Klassen.
- Sie können nicht direkt instanziiert werden.
- Verwendung des 'abc'-Moduls in Python.

Warum abstrakte Klassen verwenden?

- Erzwingen die Implementierung bestimmter Methoden in Unterklassen.
- Stellen eine konsistente Schnittstelle über verschiedene Unterklassen sicher.

Beispiele für den Einsatz:

- Formen wie Rechtecke und Kreise.
- Fahrzeugtypen wie Autos und Motorräder.

Beispiel für abstrakte Klassen

Codebeispiel:

```
1 from abc import ABC, abstractmethod
2
3 class Shape(ABC):
4     @abstractmethod
5     def area(self):
6         pass
7
8     @abstractmethod
9     def perimeter(self):
10        pass
```

Beispiel für abstrakte Klassen - 2

Codebeispiel:

```
1 class Rectangle(Shape):
2     def __init__(self, width, height):
3         self.width = width
4         self.height = height
5
6     def area(self):
7         return self.width * self.height
8
9     def perimeter(self):
10        return 2 * (self.width + self.height)
```

Übung:

- Implementieren Sie eine 'Circle'-Klasse, die von 'Shape' erbt.
- Fügen Sie die Methoden 'area' und 'perimeter' hinzu.

Statische und Klassenmethoden

Definitionen:

- **Statische Methoden:** Definiert mit '@staticmethod'. Sie hängen nicht von Instanz- oder Klassenvariablen ab.
- **Klassenmethoden:** Definiert mit '@classmethod'. Sie nehmen die Klasse als ersten Parameter.

Anwendungsfälle:

- Statische Methoden für Hilfsfunktionen (z.B. Berechnungen).
- Klassenmethoden für Factory-Methoden.

Beispiel für statische und Klassenmethoden

Codebeispiel:

```
1 class MathUtils:
2     @staticmethod
3     def add(a, b):
4         return a + b
5
6     @classmethod
7     def pi(cls):
8         return 3.14159
9
10 print(MathUtils.add(5, 7)) # 12
11 print(MathUtils.pi())     # 3.14159
```

Diskussion:

- Wann sollte eine statische Methode anstelle einer Instanzmethode verwendet werden?
- Wie unterscheiden sich Klassenmethoden von statischen Methoden?

Diskussion:

- Wann sollte eine statische Methode anstelle einer Instanzmethode verwendet werden?
 - Statische Methoden sind sinnvoll, wenn die Methode keine Instanz- oder Klassenvariablen benötigt.
- Wie unterscheiden sich Klassenmethoden von statischen Methoden?
 - Klassenmethoden erhalten automatisch die Klassenreferenz 'cls' und können Klassenvariablen verändern.

Fazit

Wichtige Erkenntnisse:

- Kapselung schützt Daten innerhalb einer Klasse.
- Abstrakte Klassen bieten eine Blaupause für konsistente Schnittstellen.
- Statische und Klassenmethoden ermöglichen Dienstprogramme und klassenbezogene Verhaltensweisen.
- OOP hilft effektiv bei der Modellierung realer Probleme.

Nächste Schritte:

- Modellierung realer Probleme üben.
- Heutige Codebeispiele überprüfen und erweitern.
- Experimentieren Sie mit abstrakten Klassen und Methoden.