Polymorphismus in Python

Definition: Polymorphismus bedeutet viele Formen und bezieht sich auf die Fähigkeit, eine Methode auf unterschiedliche Weise zu verwenden, je nach dem Objekt, das sie aufruft.

Zweck: Ermöglicht es Funktionen und Methoden, auf Objekte unterschiedlicher Typen zu reagieren, was den Code flexibler, erweiterbarer und leichter wartbar macht.

Arten von Polymorphismus

- **Kompilierungszeit-Polymorphismus**: Auch bekannt als statischer Polymorphismus. Beispiele sind Methodenüberladung (method overloading) und Operatorüberladung (operator overloading).
  - *Wie andere Sprachen (wie Java, C++) unterstützt Python standardmäßig keine Methodenüberladung. Es gibt jedoch verschiedene Möglichkeiten, Methodenüberladung in Python zu erreichen (durch optionale Parameter, *args, **kwargs oder @singledispatchmethod).*
  - Python implementiert Operatorüberladung durch die magischen Methoden - z. B. `__add__` für den **+** Operator.
- **Laufzeit-Polymorphismus**: Auch bekannt als dynamischer Polymorphismus. Wird durch Methodenüberschreibung (method overriding) erreicht.

Polymorphismus in Python

Code-Beispiel - Erreichen der Methodenüberladung in Python

```
class Beispiel:
    @staticmethod
    def anzeigen(a=None, b=None):
        if a is not None and b is not None:
            print("Methode Reaktion Typ 1")
        elif a is not None:
            print("Methode Reaktion Typ 2")
        elif b is not None:
            print("Methode Reaktion Typ 3")
        else:
            print("Methode Reaktion Typ 4")


Beispiel.anzeigen(10, 20)  #Methode Reaktion Typ 1
Beispiel.anzeigen(10)  # Methode Reaktion Typ 2
Beispiel.anzeigen(b=6)  # Methode Reaktion Typ 3
Beispiel.anzeigen()  # Methode Reaktion Typ 4
```

Polymorphismus in Python

Code-Beispiel - Methodenüberschreibung in Python

```
class Essen:
    def geschmack(self):
        return "Dieses Essen hat einen allgemeinen Geschmack."

class Pizza(Essen):
    def geschmack(self):
        return "Diese Pizza schmeckt würzig und käsig."

class Eis(Essen):
    def geschmack(self):
        return "Dieses Eis schmeckt süß und cremig."

class Kaffee(Essen):
    def geschmack(self):
        return "Dieser Kaffee schmeckt bitter und aromatisch."
```

```
# Liste von verschiedenen Essensobjekten
essen_liste = [Pizza(), Eis(), Kaffee()]

# Iteration durch die Essensliste und Ausgabe des Geschmacks
for essen in essen_liste:
    print(essen.geschmack())

# Ausgabe
```

Diese Pizza schmeckt würzig und käsig.
Dieses Eis schmeckt süß und cremig.
Dieser Kaffee schmeckt bitter und aromatisch.

Wir können sehen, dass die Methode geschmack() je nach Objekt unterschiedlich reagiert. Dies zeigt, wie Polymorphismus es ermöglicht, dass eine Methode abhängig vom Objekt **unterschiedliche Ergebnisse liefert**, was den Code flexibler und anpassungsfähiger macht.

Polymorphismus in Python

Beispiele aus der Praxis

- **GUI-Frameworks**: Grafische Benutzeroberflächen (GUI) bestehen aus vielen verschiedenen Komponenten wie Buttons, Textfeldern und Menüs. Diese Komponenten reagieren auf Benutzerinteraktionen wie Klicks und Tastatureingaben.
- **Datei-I/O**: Verschiedene Dateitypen (Text, Binär) können mit derselben Schnittstelle verarbeitet werden.
- **Spieleentwicklung:** In der Spieleentwicklung gibt es verschiedene Spielobjekte wie Spieler, Gegner und Hindernisse, die alle auf die Methode update reagieren.
- **Finanzanwendungen:** In Finanzanwendungen können verschiedene Kontotypen (Spar-, Girokonto) durch eine gemeinsame Schnittstelle verarbeitet werden.

Polymorphismus in Python

Mehr erkunden:

https://medium.com/@codingcampus/polymorphism-in-python-with-examples-887e2d45327a

https://www.programiz.com/python-programming/polymorphism

https://www.geeksforgeeks.org/polymorphism-in-python/