Scopes und Seiteneffekte in Python

Einführung

In Python bezieht sich der Scope auf den Bereich im Code, in dem Variablen sowohl zugänglich als auch änderbar sind. Verständnis von Scopes hilft bei der Vermeidung von Konflikten in Namen und bei der Kontrolle der Zugänglichkeit von Variablen. Seiteneffekte treten auf, wenn eine Funktion oder ein Ausdruck den Zustand außerhalb seines lokalen Scopes ändert, was die Vorhersagbarkeit und Wiederverwendbarkeit des Codes beeinträchtigen kann.

Scopes in Python

Lokaler Scope

Ein lokaler Scope bezieht sich auf Variablen, die innerhalb einer Funktion definiert sind und nur dort zugänglich sind.

```
def funktion():
    lokal_var = "Ich bin lokal"
    print(lokal_var)

funktion()
# Zugriff auf lokal_var von außerhalb der Funktion führt zu einem Fehler
```

Umschließender (Enclosing) Scope

Bezieht sich auf den Scope von umschließenden Funktionen, in dem eine innere Funktion auf Variablen der äußeren Funktion zugreifen kann.

```
def äußere():
    umschließende_var = "Ich bin umschließend"

    def innere():
        print(umschließende_var)

    innere()

äußere()
```

Globaler Scope

Globale Variablen sind im gesamten Code zugänglich, da sie außerhalb aller Funktionen definiert sind.

```
globale_var = "Ich bin global"

def funktion():
    print(globale_var)

funktion()
```

Eingebauter Scope

Bezieht sich auf Namen, die in Pythons eingebauten Modulen vordefiniert sind, wie `len`, `range` etc.

```
print(len("Ich bin eingebaut"))
```

Seiteneffekte

Ein Seiteneffekt tritt auf, wenn eine Funktion den Zustand außerhalb ihres lokalen Scopes ändert, z.B. durch Ändern einer globalen Variablen oder Modifikation eines übergebenen Arguments.

Beispiel für einen Seiteneffekt

```
globale_liste = []

def füge_hinzu(element):
    globale_liste.append(element)

füge_hinzu(1)
print(globale_liste)  # [1]
```

Vermeiden von Seiteneffekten

Die Verwendung von reinen Funktionen, die keine externen Zustände ändern und die gleichen Ergebnisse für die gleichen Eingaben liefern, hilft, Seiteneffekte zu vermeiden.

Beispiel einer reinen Funktion

```
def addiere(a, b):
    return a + b

resultat = addiere(1, 2)
print(resultat)  # 3
```