LU02d - By Value und By Reference in Python

Einführung

In Python ist es wichtig zu verstehen, wie Daten zwischen Funktionen und Variablen übergeben werden. Python verwendet sowohl By `Value` als auch By `Reference` Mechanismen, abhängig vom Datentyp. Das Verständnis dieser Konzepte hilft, den Programmfluss und das Verhalten von Variablen in verschiedenen Kontexten besser zu verstehen.

By Value

By `Value` bedeutet, dass der Wert einer Variable kopiert wird, wenn er an eine Funktion übergeben wird. Änderungen am kopierten Wert haben keine Auswirkungen auf die ursprüngliche Variable. In Python gilt dieses Konzept hauptsächlich für unveränderliche (immutable) Datentypen.

Beispiele für Immutable Datentypen

Zu den unveränderlichen Datentypen in Python gehören:

- int (z.B. `5`)
- float (z.B. `3.14`)
- str (z.B. `'Hallo'`)
- tuple (z.B. `(1, 2, 3)`)

Wenn eine Variable mit einem dieser Typen an eine Funktion übergeben wird, wird eine Kopie des Wertes erstellt. Jede Änderung innerhalb der Funktion betrifft nur die Kopie, nicht die ursprüngliche Variable.

Beispiel 1: By Value mit Immutable Typen

```
def modify_value(x):
    """
    Tries to modify the immutable value by adding 10.
    """
    x += 10
    return x


if name == 'main':
    a = 5
    new_value = modify_value(a)
    print(f'Originaler Wert: {a}') # Output: Originaler Wert: 5
    print(f'Neuer Wert: {new_value}') # Output: Neuer Wert: 15
```

Hier bleibt die ursprüngliche Variable a unverändert, da `int` ein unveränderlicher Typ ist. Die Änderung findet nur in der Kopie innerhalb der Funktion statt, und der ursprüngliche Wert bleibt erhalten.

By Reference

By `Reference` bedeutet, dass eine Referenz auf die Originaldaten übergeben wird, anstatt eine Kopie zu erstellen. Änderungen an diesen Daten innerhalb der Funktion beeinflussen somit die ursprüngliche Variable. In Python gilt dieses Konzept hauptsächlich für veränderbare (mutable) Datentypen.

Beispiele für Mutable Datentypen

Zu den veränderbaren Datentypen in Python gehören:

- list (z.B. `[1, 2, 3]`)
- dict (z.B. `{'a': 1, 'b': 2}`)
- set (z.B. `{1, 2, 3}`)

Wenn eine Variable mit einem dieser Typen an eine Funktion übergeben wird, wird eine Referenz auf das Originalobjekt übergeben. Jede Änderung innerhalb der Funktion wirkt sich direkt auf das ursprüngliche Objekt aus.

Beispiel 2: By Reference mit Mutable Typen

```
def modify_list(some_list):
    """
    Appends the value 4 to the passed list.
    """
    some_list.append(4)


if name == 'main':
    my_list = [1, 2, 3]
    modify_list(my_list)
    print(f'Geänderte Liste: {my_list}') # Output: Geänderte Liste: [1, 2,
3, 4]
```

In diesem Beispiel wird die Liste `my_list` direkt innerhalb der Funktion geändert. Da Listen in Python mutable sind, wird die Original-Liste durch die Änderung in der Funktion beeinflusst.

Unterschiede und Auswirkungen

Die Art und Weise, wie Werte und Referenzen in Python übergeben werden, hat direkte Auswirkungen auf das Verhalten und die Vorhersagbarkeit des Codes:

- **Immutable Objekte:** Änderungen in einer Funktion betreffen nur die lokale Kopie. Der Originalwert bleibt unverändert.
- **Mutable Objekte:** Änderungen in einer Funktion betreffen direkt das Originalobjekt. Dies kann zu unbeabsichtigten Seiteneffekten führen, wenn die Funktion den Zustand des Objekts verändert.

Um ungewollte Seiteneffekte zu vermeiden, sollten Sie sich der Datenarten bewusst sein, die Sie in Funktionen verwenden, und überlegen, ob das Objekt unverändert bleiben soll oder nicht.

Zusammenfassung

Das Verständnis von By `Value` und By `Reference` ist entscheidend für das Schreiben von klarem und wartbarem Code in Python. Immutable Objekte bieten Schutz vor unbeabsichtigten Änderungen, während mutable Objekte Flexibilität bieten, aber sorgfältig gehandhabt werden müssen, um unvorhergesehene Probleme zu vermeiden.

Tipp: Wenn Sie mutable Objekte verwenden, aber unerwartete Änderungen vermeiden möchten, sollten Sie in Erwägung ziehen, eine Kopie des Objekts zu erstellen, bevor Sie es an eine Funktion übergeben.

M323-LU02, M323-AE1