LU02b - Pure Functions

Definition und Bedeutung von Pure Functions

Pure Functions, auch reine Funktionen genannt, sind ein Grundkonzept der funktionalen Programmierung. Sie haben zwei Hauptmerkmale:

1. **Deterministisch**: Das Ergebnis einer pure function hängt ausschließlich von den übergebenen Eingabewerten ab und hat keine Seiteneffekte. Das bedeutet, dass sie bei gleichen Eingabewerten immer das gleiche Ergebnis liefern.
2. **Keine Seiteneffekte**: Eine pure function verändert nichts in ihrem Umfeld. Sie wirkt sich nicht auf den globalen Zustand aus und verändert keine Eingabeparameter.

Ein einfaches Beispiel in Python könnte eine Funktion zur Addition zweier Zahlen sein:

```
def add(x, y):
  return x + y
```

Warum Pure Functions? Vorteile und Anwendungsbereiche

Pure Functions bieten mehrere Vorteile:

1. **Testbarkeit**: Da das Ergebnis nur von den Eingaben abhängt, sind pure functions leicht zu testen.
2. **Wartbarkeit**: Sie sind leichter zu verstehen und zu warten, da sie keine externen Zustände verändern.
3. **Parallelisierung**: Da sie keine Seiteneffekte haben, können sie sicher parallel ausgeführt werden.

Pure Functions sind in vielen Programmierparadigmen nützlich, nicht nur in der funktionalen Programmierung. Sie können zur Verbesserung der Codequalität in jedem Kontext beitragen.

Vergleich mit Unpure Functions

Unpure Functions, im Gegensatz zu pure functions, können Seiteneffekte haben oder von externen Zuständen abhängig sein. Sie können somit unerwartete Ergebnisse liefern, was sie schwerer zu verstehen und zu warten macht.

Beispiel 1

Betrachten Sie das folgende Beispiel mit einer Variablen:

```
total = 0
```

```
def add_to_total(x):
  global total
  total += x

if __name__ == '__main__':
    add_to_total(3)
    print(total) #3
```

Diese Funktion verändert den globalen Zustand (die Variable `total`) und ist somit eine unpure function. Das `global`-Schlüsselwort signalisiert dem Entwickler und dem Python-Interpreter, dass die Funktion eine globale Variable ändert. Dies macht es einfacher, „unpure“ Funktionen zu erkennen und fördert die Verwendung von „pure“ Funktionen, indem es die Aufmerksamkeit auf die Änderung des globalen Zustands lenkt.

Beispiel 2

In diesem Beispiel haben wir eine Liste namens `my_list`, die außerhalb der Funktion `modify_list` definiert ist. Die Funktion `modify_list` ändert diese Liste.

```
# Definieren einer externen Liste
my_list = [1, 2, 3]

def modify_list():
    """
    Modify the external_list by appending a value to it.
    """
    return my_list.append(4)

if __name__ == '__main__':
    print('Vor der Änderung: ', my_list)  # Output: Vor der Änderung: [1, 2,
3]
    new_list = modify_list()
    print('Neue Liste: ', new_list)  # Output: Neue Liste: [1, 2, 3, 4]
    print('Nach der Änderung: ', my_list)  # Output: Nach der Änderung: [1,
2, 3, 4]
```

Wie Sie sehen können, ändert die Funktion `modify_list` den Zustand der `my_list`. Daher ist `modify_list` eine unpure Funktion. In diesem Fall wird das `global`-Schlüsselwort nicht benötigt, da Listen in Python „mutable“ (veränderbar) sind. Das heißt, sie können innerhalb einer Funktion verändert werden, ohne dass das `global`-Schlüsselwort benötigt wird. Das kann zu einer weniger klaren Code-Struktur führen, da es schwieriger ist, solche Änderungen auf den ersten Blick zu erkennen.

Beispiel 3

In diesem Beispiel wird eine Liste als Argument an eine Funktion übergeben. Die Funktion modifiziert die Liste direkt, was zeigt, dass Listen in Python by reference übergeben werden.

```
def append_to_list(some_list, append):
    """
    Appends the value ''append'' to the passed list.
    """
    some_list.append(append)

if name == 'main':
    my_list = [1, 2, 3]
    print('Vor der Änderung:', my_list) # Output: Vor der Änderung: [1, 2,
3]
    append_to_list(my_list,5)
    print('Nach der Änderung:', my_list) # Output: Nach der Änderung: [1, 2,
3, 5]
```

In diesem Fall sehen wir, dass die Funktion `append_to_list` die Liste `my_list` ändert, indem sie ein weiteres Element hinzufügt. Obwohl `my_list` als Argument übergeben wird, wird es innerhalb der Funktion verändert, ohne dass es zurückgegeben werden muss. Das zeigt, dass die Liste by reference übergeben wird und Änderungen an der Liste in der Funktion Auswirkungen auf die ursprüngliche Liste außerhalb der Funktion haben.

Dieses Verhalten macht `append_to_list` ebenfalls zu einer unpure function, da sie den Zustand der übergebenen Liste direkt verändert.

Wichtiger Hinweis: Dieses Beispiel verdeutlicht die Implikationen der Übergabe von mutable Objekten wie Listen in Python. Es ist wichtig zu verstehen, dass solche Änderungen schwer nachvollziehbar sein können, insbesondere in komplexeren Programmen, und das Verständnis des Unterschieds zwischen pure und unpure functions erleichtert es, vorhersehbaren und wartbaren Code zu schreiben.

Fazit

Pure Functions sind ein mächtiges Konzept, das nicht nur die Qualität des Codes verbessert, sondern auch zur Entwicklung von effizienteren, wartbareren und sichereren Programmen beiträgt. Die Unterscheidung zwischen pure und unpure functions hilft, die Prinzipien der funktionalen Programmierung besser zu verstehen und in der Praxis umzusetzen.

M323-LU02, M323-AG1