Comprehensions

Eric Niklas Wolf, Moritz Pflügner

Python-Kurs

7. Dezember 2021

Gliederung

Basics

Comprehensions sind eine bequeme Art und Weise, um Funktoren (Datenstrukturen, die andere Datenstrukturen beinhalten) mit kleinen Expressions zu erstellen und zu füllen und sind in allen modernen Sprachen vorhanden.

List Comprehension

Grundlegender Syntax: `[ EXPRESSION for LAUFVARIABLE in ITERABLE (if FILTER) ]`

EXPRESSION Ist ein beliebiger Ausdruck (man stelle sich ein implizites `return` vor), etwa ein Wert, eine Variable, eine Gleichung, etc ...
`EXPRESSION` wird am Ende in der Liste abgelegt.

LAUFVARIABLE Eine beliebige Variable, die in *EXPRESSION* und *FILTER* zur Verfügung steht

ITERABLE Ist häufig etwas wie `range()` oder eine andere Liste.

FILTER Eine optionale boolean expression, womit Einträge gefiltert werden (falls `False`). Nützlich, wenn z.B. nur gerade Zahlen übernommen werden sollen, usw...

List Comprehension - Beispiel

```
1 varList = [var*8 for var in range(10)]
2 # => [0, 8, 16, 24, 32, 40, 48, 56, 64, 72]
3
4
5 # Mit Filter (hier: Fuer i ist gerade)
6 evenVarList = [var*8 for var in range(10) if var % 2 == 0]
7 # => [0, 16, 32, 48, 64]
```

Dict Comprehension

Grundlegender Syntax: `{ KEY : VALUE for LAUFVARIABLE in ITERABLE (if FILTER) }`

Fast der gleiche Syntax, nur dieses Mal mit 2 Expressions: *KEY* und *VALUE*. Ansonsten gelten die gleichen Regeln.

Dict Comprehension - Beispiel

```
1 liste = ["Fritz", "Alex", "Nadine", "Peter", "Anna"]
2
3 names = {key: len(key) for key in liste}
4 # => {'Peter': 5, 'Fritz': 5, 'Alex': 4, 'Anna': 4, 'Nadine
      ': 6}
```

Generators

Generator Ein Objekt, über das iteriert werden kann. Wenn ein Element daraus verwendet wurde, ist es nicht mehr in dem Generatorobjekt enthalten.

Grundlegender Syntax: `( EXPRESSION for LAUFVARIABLE in ITERABLE (if FILTER) )`
Da sich `list` und `dict` auch aus Iterables bauen lassen, gilt prinzipiell:

```
list(EXPRESSION for VARIABLE in ITERABLE) == [EXPRESSION for VARIABLE in ITERABLE]
```

und

```
dict((KEY, VALUE) for VARIABLE in ITERABLE) == {KEY:VALUE for VARIABLE in ITERABLE}
```

Aber: Generatoren sind lazy, sie erzeugen die Elemente erst wenn sie iteriert werden.

Generators - Beispiel

```
1  # liefert gerade Zahlen von 0 bis 10 (10 nicht enthalten)
2  generator = (i for i in range(10) if i % 2 == 0)
3
4  # gibt 0, 2, 4, 6 und 8 aus
5  for number in generator:
6      print(number)
7
8  # gibt nichts aus, generator ist erschoepft
9  for number in generator:
10     print(number)
11
12 # wenn alle Elemente sofort erzeugt werden wuerde mindestens
       4GB Speicher benoetigt
13 for number in (i for i in range(2**32)):
14     print(number)
```

Nesting

`for` Schleifen in Comprehensions können verschachtelt werden. Dabei werden sie von Links nach Rechts ausgeführt, was man bei Variablen beachten muss.
Wichtig: Starke Verschachtelung verringert die Lesbarkeit!

Nesting - Beispiel

```
1  # erzeugt eine Liste welche jede Zahl n
2  # von 0 bis 4 (4 nicht enthalten) n mal enthaelt
3  list1 = [i for i in range(4) for _ in range(i)]
4
5  # gleicher Code ohne Comprehension
6  list2 = []
7  for i in range(4):
8      for _ in range(i):
9          l.append(i)
10
11 list1 == list2 == [1,2,2,3,3,3]
12
13 # loest einen NameError aus, weil
14 # a erst durch die zweiten Schleife entsteht
15 [i for i in range(a) for a in range(i)]
16
17 # zaehlt fuer jede Zahl n von 0 bis 4 (4 nicht enthalten)
18 # von 0 bis n-1, weil EXPRESSION erst nach der letzten
19 # Schleife evaluiert wird
20 list3 = [a for i in range(4) for a in range(i)]
21
22 list3 == [0,0,1,0,1,2]
```