Skriptsprachen: Python
Erweiterte Datenstrukturen

Jan Krüger, Alexander Sczyrba

Technische Fakultät
Universität Bielefeld

12. Februar 2019

Python

–

Listenbeschreibung

range(), Generatoren, Listbeschreibungen

range()

range([[start, [end]], step])) Erzeugt eine Liste von *start*bis *end* mit Schrittweite *step*

- Rückgabewert: `range`-Objekt, keine Liste
- Ist *iterable*
- Wird in Schleifen verwendet und ist speicheroptimiert (Generator)
- Besonders performant, wenn nicht alle Elemente aufgerufen werden (lazy evaluation)

Generatoren

Wie sieht das Erstellen einer Liste mit genau diesem (y) Inhalt aus?

```
> retL = []
> for y in range(2,4):
>     retL.append(y/2.)
> retL
[1.0, 1.5]
```

Geht das nicht auch effizienter? JA!

```
> retLfast = []
> g = ( y/2. for y in range(2, 4))
> for y in g:
>    retLfast.append(y)
> retL
[1.0, 1.5]
```

Generatoren

Der Ausdruck `( y/2. for y in range2, 4))` wird *Generator* genannt.

- Definierte Rechenoperation wird nur ausgeführt, wenn wir einen weiteren Schleifendurchlauf haben
- Speichereffizient, weil nicht die gesamte Liste generiert wird
- Wenn ein `break` auftritt, kann an anderer Stelle ab dem nächsten Element weitergearbeitet werden

In unserem Fall ist die Listenerzeugung dennoch nicht schnell und elegant genug.

Listenbeschreibung

Der Generatorausdruck kann fast 1 : 1 verwendet werden, um eine Liste direkt zu erzeugen:

```
> [y/2. for y in range(2,4)]
[1.0, 1.5]
```

Dasselbe gilt auch für die Erzeugung von `dict` und `set`:

```
> #dict
> {y:y/2. for y in range2,4)}
{2: 1.0, 3: 1.5}


> #set
> {y/2. for y in range(2,4)}
{1.0, 1.5}
```

Listenbeschreibung

Was für eine Liste wird hier erzeugt?

```
> [ letter*N for N in range(1,5) for letter in 'abcd' ]
```

Listenbeschreibung

```
> [ letter*N for N in range(1,5) for letter in 'abcd' ]
['a',
 'b',
 'c',
 'd',
 'aa',
 'bb',
 'cc',
 'dd',
 'aaa',
 'bbb',
 'ccc',
 'ddd',
 'aaaa',
 'bbbb',
 'cccc',
 'dddd']
```

Übung – FASTA-Listenbeschreibung

Überlegt euch eine Listenbeschreibung für eine FASTA-Sequenz, bei der die resultierende Liste so aussehen könnte:
`["A", "A", "C", "T", "A", "G"]`

Erstellt anschließend eine Funktion `generate_fasta`, welche eine zufaellige FASTA Sequenz mit Hilfe der Listenbeschreibung generiert. Die Funktion erwartet die Länge der Sequenz und die Anzahl der Zeichen pro Zeile (optional, default:20) als Parameter. Der Rückgabewert ist ein formatierter Fasta-String.

Hinweis: Das Modul `random` enthaelt Funktionen um Zufallszahlen zu erzeugen. Nutzt die integrierte Hilfe von `ipython` um eine geeignete Funktion zu finden ...

Wahrheitgenerator

Das Folgende kleine Skriptlet verwendet einen besonders ausgefeilten Listengenerator, der euch die Wahrheit näher bringt.

```
> [ '%s␣is␣%s' % (item, bool(item)) for item in
   [0, 1, 0.0, 0.1, 0j, 1j, [], [0], (), (0,), {}, {'a':'A'}, None]]

['0␣is␣False',
 '1␣is␣True',
 '0.0␣is␣False',
 '0.1␣is␣True',
 '0j␣is␣False',
 '1j␣is␣True',
 '[]␣is␣False',
 '[0]␣is␣True',
 '()␣is␣False',
 '(0,)␣is␣True',
 '{}␣is␣False',
 "{'a':␣'A'}␣is␣True",
 'None␣is␣False']
```

Python

–

Funktionen höherer Ordnung

map(), filter()

Funktionen höherer Ordnung

Funktionen höherer Ordnung (high order functions) sind Funktionen, die Funktionen als Argumente erhalten können und wiederum Funktionen als Rückgabewerte haben können.

Beispiel aus Haskell:

```
> map :: (a -> b) -> [a] -> [b]
> map f []     = []
> map f (x:xs) = (f x):map f xs
```

`map (\x -> x^2) [1,2,3,4]` wird ausgewertet zu `[1,4,9,16]`

Aber wie sieht das nun in Python aus? Kurz zur Erinnerung: In Python ist selbst eine Funktion ein Objekt!

map()

`map(<FUNC>, <SEQ>[, <SEQ>, ...])` erwartet als Argumente genau eine Funktion und mindestens eine Sequenz. Werden mehr Sequenzen als eine Sequenz angeben, so werden diese als weitere Argumente an die Funktionen übergeben. Stimmen die Längen nicht überein, werden die Elemente entsprechend `None` gesetzt.

```
> def pow_2(n, e=2):
>  return n**e

> map(pow_2, range(1,5))
[1, 4, 9, 16]
> map(pow_2, range(1,5), range(2,6))
[1, 8, 81, 1024]
> map(lambda n:n**2, range(1,5))
[1, 4, 9, 16]
```

Übung – `map()` mit `lambda`

Forme den `lambda`-Ausdruck in `map()` von der vorherigen Folie entsprechend um, sodass die Funktionalität der `pow_2(n, e=2)`-Funktion entspricht.

Eigene Funktionen höherer Ordnung

Es lassen sich auch eigene Funktionen höherer Ordnung in Python schreiben. Hier ein kleines Beispiel für $f(x) = x + 3$ und $g(x) = f(f(x))$:

```
> def f(x):
>   return x + 3

> def twice(function, x):
>   return function(function(x))

> print(twice(f, 7))
```

Wie sähe hier das Ergebnis aus?

filter()

`filter(<FUNC> | None, <SEQ>)` erzeugt eine Liste, bei der die Anwendung von `FUNC` `True` ergibt. Ist `FUNC = None`, so werden nur die Elemente in die neue Sequenz übernommen, die `True` sind. Wenn `SEQ` ein Tuple oder ein String ist, so bleibt der Rückgabetyp entsprechend des Eingabetyps, ansonsten wird immer eine Liste zurückgegeben.

```
> def is_odd(n):
>  return n%2

> filter(is_odd, range(4))
[1, 3]
> filter(None, range(4)
[1, 2, 3]
```

Python

–

Vertiefung sort()

key, reverse

sort()

Bisher habt ihr `sort()` nur mit den Standardwerten verwendet, dabei ist die Funktionen noch viel mächtiger. Kurze Erinnerung `sort([key[, reverse]])`:

key
: Funktion, die definiert, nach welcher Eigenschaft der Sequenzelemente sortiert werden soll (z.B. `key=str.lower`)

reverse
: Wenn `True`, wird die Sortierreihenfolge umgekehrt

sort()

```
> mydict= { 'c':25, 'a':15,  'd':7, 'b':3 }

 > sorted(mydict.items())
 [('a', 15),
  ('b', 3),
  ('c', 25),
  ('d', 7)]
```

Die Funkton `sorted()` sortiert alle *iterable* Datentypen und liefert eine Liste mit den sortierten Element zurück. Hier sehen wir, dass nach den *Keys* sortiert wird. (Warum ?)

sort()

Wie können wir die Sortierung umkehren? → `reverse=True`

```
> sorted(mydict.items(), reverse=True)
  [('d', 7),
   ('c', 25),
   ('b', 3),
   ('a', 15)]
```

Übung – sort()

Gegeben sei folgendes Dictionary :

```
> mydict2 = { 'a' : (15,24),
              'b' : (1,3),
              'c' : (6,25),
              'd' : (9,7) }
```

Nehmen wir an, dass das Value-Tupel einen Bruch definiert. Wie können wir nach diesem sortieren?