Verändern einer Sequenz beim Iterieren

POS

Dr. Günter Kolousek

2016

Creative Commons Namensnennung - Nicht kommerziell - Keine Bearbeitungen 4.0 International Lizenz

1 Problematik

Betrachten wir die Aufgabenstellung, dass aus einer Liste von Wörtern alle Wörter mit einer Länge kleiner oder gleich 2 gelöscht werden sollen.

Der naive Ansatz sieht folgendermaßen aus:

```
lst = ["abc", "def", "xy", "ghi", "jkl"]

i = 0
for w in lst:
    if len(w) <= 2:
        del lst[i]
    i += 1

print(lst)
```

Damit kommt es zu folgender erwarteter Ausgabe:

```
['abc', 'def', 'ghi', 'jkl']
```

Ändert man die Liste in diesem Programm allerdings ab, dann erhält man **nicht** mehr das erwartete Ergebnis:

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

i = 0
for w in lst:
    if len(w) <= 2:
        del lst[i]
    i += 1

print(lst)
```

Das Ergebnis enthält noch immer den String ='ab'=:

```
['abc', 'def', 'ab', 'ghi', 'jkl']
```

Woran liegt das? Das liegt daran, dass der Index um eins weitergestellt wird, aber im gleichen Schritt das Element aus der Liste gelöscht wird. Damit wird das zweite kurze Wort einfach übersprungen.

2 Lösung mit Kopie

Eine einfache Lösung besteht darin, dass man das Iterieren über die Sequenz von den Löschoperationen trennt. Das kann man dadurch erreichen, dass man eine Kopie der Liste anlegt und über die Kopie iteriert und die Löschoperationen in dem Originalobjekt durchführt.

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

lst2 = lst
i = 0
for w in lst2:
    if len(w) <= 2:
        del lst[i]
    i += 1

print(lst)
```

Das Ergebnis ist allerdings in dieser Form ebenfalls **nicht** richtig:

```
['abc', 'def', 'ab', 'ghi', 'jkl']
```

Das funktioniert so nicht, da wir keine Kopie des Objektes sondern nur einen neuen Namen angelegt haben. Wir benötigen eine Kopie!

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

lst2 = lst.copy()  # creates a shallow copy
i = 0
for w in lst2:
    if len(w) <= 2:
        del lst[i]
    i += 1

print(lst)
```

Allerdings führt auch dies alleine **nicht** zum gewünschten Ergebnis:

```
['abc', 'def', 'ab', 'jkl']
```

In diesem Fall liegt das Problem noch immer daran, dass wir weiterzählen, obwohl wir gar nicht weiterzählen dürften, wenn wir ein Element herauslöschen:

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

lst2 = lst.copy()
i = 0
for w in lst2:
    if len(w) <= 2:
        del lst[i]
    else:
        i += 1

print(lst)
```

Damit kommt es zu der korrekten Ausgabe:

```
['abc', 'def', 'ghi', 'jkl']
```

Diese Lösung hat den prinzipiellen Vorteil, dass diese auch im Kontext von nebenläufigen Operationen auf der selben Datenstruktur funktioniert (solange die Kopie thread-safe erstellt wurde).

3 Lösung mit korrekten Indizes

Obwohl die vorhergehende Lösung funktioniert, hat diese den Nachteil, dass eine Kopie der ursprünglichen Liste erstellt wird. Das Erstellen als auch das Entfernen der Kopie benötigt Zeit und die Kopie selber bedingt einen zusätzlichen Speicherbedarf. Daher ist eine Lösung vorzuziehen, die *ohne* eine Kopie der Liste auskommt.

Nachfolgend eine Lösung basierend auf eine `while`-Schleife. Um die Funktion dieser Schleife besser verstehen zu können, habe ich noch eine zusätzliche Ausgabe eingefügt:

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

i = 0
j = 0
max_idx = len(lst) - 1
while True:
    print(i, lst[i])
    if len(lst[i]) <= 2:
        del lst[i]
    else:
        i += 1
    if j == max_idx:
        break
    j += 1

print(lst)
```

Hier wiederum die Ausgabe:

```
0 abc
1 def
2 xy
2 ab
2 ghi
3 jkl
['abc', 'def', 'ghi', 'jkl']
```

Allerdings ist die Verwendung einer ”Endlosschleife” mit ”break” nicht sonderlich elegant. Man kann diese leicht in eine range-basierte Schleife umformen:

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

i = 0
for j in range(len(lst)):
    if len(lst[i]) <= 2:
        del lst[i]
    else:
        i += 1

print(lst)
```

Hier die Ausgabe:

```
['abc', 'def', 'ghi', 'jkl']
```

Diese Lösung benötigt noch immer eine zusätzliche Variable (”i”).

4 Lösung mit Iterieren vom Ende

Um sich auch noch die zusätzliche Variable zu ersparen kann man die Iteration vom Ende weg zum Anfang hin vornehmen. Damit erspart man sich auch die Abfrage, ob weitergezählt werden soll oder nicht:

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

i = len(lst) - 1
for w in lst[::-1]:
    if len(w) <= 2:
        del lst[i]
    i -= 1

print(lst)
```

Und hier wiederum die Ausgabe:

```
['abc', 'def', 'ghi', 'jkl']
```

Mittels `lst[::-1]` kann die Liste umgekehrt werden und damit gegensinnig durchlaufen werden. Allerdings wird damit ebenfalls eine zusätzliche Liste erstellt, die man einfach vermeiden kann:

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

i = len(lst) - 1
while i >= 0:
    if len(lst[i]) <= 2:
        del lst[i]
    i -= 1

print(lst)
```

Nochmals die korrekte Ausgabe:

```
['abc', 'def', 'ghi', 'jkl']
```

Verwendet man allerdings die Funktion `range` dann wird keine Liste erstellt. `range` liefert ein Objekt zurück, das einen ”faulen” Datentyp darstellt und nicht eine Liste! `range` kann leicht in einer Schleife verwendet werden, wodurch die Lösung nochmals einfacher wird:

```
lst = ["abc", "def", "xy", "ab", "ghi", "jkl"]

for i in range(len(lst) - 1, -1, -1):
    if len(lst[i]) <= 2:
        del lst[i]

print(lst)
```

Ausgabe:

```
['abc', 'def', 'ghi', 'jkl']
```