Solution Alternatives Training Exercise 3.4

Claus Führer
Center for Mathematical Sciences
Lund University
November 9, 2017

1 Preparation

Here is the given distance table in Python. Could be the distances between towns in Skåne.

In [1]: distance=[[0,20,30,40],
              [20,0,50,60],
              [30,50,0,70],
              [40,60,70,0]]
   print(distance)

[[0, 20, 30, 40], [20, 0, 50, 60], [30, 50, 0, 70], [40, 60, 70, 0]]

The task is to reduce it to the lower part, i.e. to remove the diagonal and the upper part, which contains redundant information.

2 First solution

In [2]: n=len(distance)

reddistance=[] # initialize by an empty list

for i in range (n): #i is the row counter
    row=distance[i]
    redrow=[] # initialize an empty list for the reduced row
    for j in range(n): #j is the row counter
        if row [j] == 0 :
            break
        redrow.append(row[j])
    reddistance.append(redrow)

print(reddistance)

[[], [20], [30, 50], [40, 60, 70]]
If you do not like the empty list at the beginning, start the outer for loop a step later:

```python
In [3]: n=len(distance)
   : reddistance=[]
   :
   : for i in range (1,n):  # we started with the second row instead (index 1!)
   :     row=distance[i]
   :     redrow=[]
   :     for j in range(n):
   :         if row [j] == 0 :
   :             break
   :         redrow.append(row[j])
   :     reddistance.append(redrow)
   : print(reddistance)
[[20], [30, 50], [40, 60, 70]]
```

This first solution works with counters (i, j). The second object works with list elements directly.

### 3 Second Solution

```python
In [4]: reddistance=[]
   : for row in distance:
   :     redrow=[]
   :     for col in row:
   :         if col==0:
   :             break
   :         redrow.append(col)
   :     reddistance.append(redrow)
   : print(reddistance)
[[], [20], [30, 50], [40, 60, 70]]
```

To avoid the empty list in the first row of the result, we might use slices:

```python
In [5]: reddistance=[]
   : for row in distance[1:]:
   :     redrow=[]
   :     for col in row:
```
if col==0:
break
redrow.append(col)
reddistance.append(redrow)

print(reddistance)

[[20], [30, 50], [40, 60, 70]]

4 Third Solution

The elegant (and computational faster way) is to avoid append and for loops by using list comprehension. Here, we will use also the enumerate command, which gives us a counter value and an object (i and row in the code below):

In [6]: reddistance=[row[0:i] for i, row in enumerate(distance)]
print(reddistance)

[[], [20], [30, 50], [40, 60, 70]]

Again, you might want to avoid the empty list. This can be done by a simple if condition in the list comprehension expression

In [7]: reddistance=[row[0:i] for i, row in enumerate(distance) if i > 0]
print(reddistance)

[[20], [30, 50], [40, 60, 70]]

5 Summary

To sum up: There are several alternativ solutions to this task. Which one was yours? Is there an advantage to avoid numbering (Second Solution) or do you prefer the First Solution? Did you remember list comprehension? Do you find the result more readable or are you in favor of for loops? In a later lecture we will check, which of the solutions is fastest to be executed.