Python join two nested lists

I have two nested lists of strings:

```
listA = [["SomeString1", "A", "1"],
         ["SomeString2", "A", "2"],
         ["SomeString3", "B", "1"],
         ["SomeString4", "B", "2"]]


listB = [["OtherString1", "A", "1"],
         ["OtherString2", "A", "2"],
         ["OtherString3", "B", "1"],
         ["OtherString4", "B", "2"]]
```

For every list in A, I want to find the list in B where `(sublistB[1] == sublistA[1]) and (sublistB[2] == sublistA[2])` (zero-indexing).

I then want to append the first entry of the 'B' sublist to the 'A' sublist, such that the final output would be:

```
joined = [["SomeString1", "A", "1", "OtherString1"],
         ["SomeString2", "A", "2", "OtherString2"],
         ["SomeString3", "B", "1", "OtherString3"],
         ["SomeString4", "B", "2", "OtherString4"]]
```

Or even better, to insert the entry to position 1:

```
joined = [["SomeString1", "OtherString1", "A", "1"],
         ["SomeString2", "OtherString2", "A", "2"],
         ["SomeString3", "OtherString3", "B", "1"],
         ["SomeString4", "OtherString4", "B", "2"]]
```

What would be the best way to do this in python? I have an implementation but with 3 nested loops and it takes some time. I have a feeling that `map`, `filter` and/or `reduce` may help, but not sure how to implement?

Note that the lists are not necessarily as neatly ordered in my example here.

Also, this is very important - the lists may not be the same length, nor is it guaranteed that every sublist contains a match. Where no match can be found, I would like to append None.

python list join filtering

edited Nov 3 '14 at 16:59
Martijn Pieters ♦
447k 56 1038 1219

asked Nov 3 '14 at 16:55
jramm
1,800 2 7 18

Does the order matter at all? A dictionary might be a better data structure, using `("A", "1")` as a key, for example. – Tim Pietzcker Nov 3 '14 at 16:57

Order does not matter. If the output is a dictionary that is ok, but the input is two lists of lists – jramm Nov 3 '14 at 17:01

2 Answers

Use a dictionary to 'index' the strings from `listB`:

```
listBstrings = {tuple(lst[1:]): lst[0] for lst in listB}
```

This maps `(listB[x][1], listB[x][2])` tuples to `listB[x][0]` strings. Now you can look these up and produce `joined` in a single loop:

```
joined = [[lst[0], listBstrings[lst[1], lst[2]]] + lst[1:] for lst in listA]
```

You may need to use `listBstrings.get((lst[1], lst[2]), '')` to produce a default empty string if the two elements were never present in `listB`.

All in all, this takes linear time O(N + M), where N and M are the input list lengths. Compare this to your nested loop approach, which takes O(N * M) quadratic time. The difference is that two lists of 10 elements each take 20 iterations with the above approach, vs. 100 in a nested loop solution, with 100 elements mine takes 200 iterations vs. nested taking 10.000 iterations,

etc.

Demo:

```
>>> from pprint import pprint
>>> listA = [["SomeString1", "A", "1"],
...          ["SomeString2", "A", "2"],
...          ["SomeString3", "B", "1"],
...          ["SomeString4", "B", "2"]]
>>> listB = [["OtherString1", "A", "1"],
...          ["OtherString2", "A", "2"],
...          ["OtherString3", "B", "1"],
...          ["OtherString4", "B", "2"]]
>>> listBstrings = {tuple(lst[1:]): lst[0] for lst in listB}
>>> joined = [[lst[0], listBstrings[lst[1], lst[2]]] + lst[1:] for lst in listA]
>>> pprint(joined)
[['SomeString1', 'OtherString1', 'A', '1'],
 ['SomeString2', 'OtherString2', 'A', '2'],
 ['SomeString3', 'OtherString3', 'B', '1'],
 ['SomeString4', 'OtherString4', 'B', '2']]
```

edited Nov 3 '14 at 17:05

answered Nov 3 '14 at 16:58

Martijn Pieters ♦

447k 56 1038 1219

Thanks, that is ok - you have reduced it to two loops which is good. – jramm Nov 3 '14 at 17:02

@jramm: it's two *sequential* loops; one over listB, the other over listA. Complexity O(N + M) where N and M are the list sizes. You used *nested* loops, resulting in O(N * M) or worse performance. – Martijn Pieters ♦ Nov 3 '14 at 17:03

This solution won't correctly handle the case where `listB` doesn't contain a corresponding row to one of the `listA` rows. It's trivial to fix: just use the `.get()` method when looking up the string in `listBstrings`, and use `None` as the default argument. – steveha Nov 3 '14 at 17:04

@jramm: in other words, my solution results in a linear time increase as you add elements to the lists, yours results in quadratic time increases. two lists of 10 elements each, processed with my solution takes 20 iterations, your nested loops take 100 iterations, etc. – Martijn Pieters ♦ Nov 3 '14 at 17:06

A similar approach to @MartijnPieters answer, but using a dict generator:

```
from pprint import pprint
listA = [["SomeString1", "A", "1"],
         ["SomeString2", "A", "2"],
         ["SomeString3", "B", "1"],
         ["SomeString4", "B", "2"],
         ["SomeString5", "C", "1"]]
listB = [["OtherString1", "A", "1"],
         ["OtherString2", "A", "2"],
         ["OtherString3", "B", "1"],
         ["OtherString4", "B", "2"],
         ["OtherString5", "C", "2"]]
dictB = dict( ((x[1], x[2]), x[0]) for x in listB )
joined = [ [ a[0], dictB.get((a[1], a[2])), a[1], a[2] ] for a in listA ]
pprint(joined)
```

The result:

```
[['SomeString1', 'OtherString1', 'A', '1'],
 ['SomeString2', 'OtherString2', 'A', '2'],
 ['SomeString3', 'OtherString3', 'B', '1'],
 ['SomeString4', 'OtherString4', 'B', '2'],
 ['SomeString5', None, 'C', '1']]
```

I'm not sure if the use of the dict generator will result in a quicker evaluation, but it would possibly save on memory use.

Another variation of this is to use two dict comprehensions and iterate of the items of one of them:

```
dictA = dict( ((x[1], x[2]), x[0]) for x in listA )
dictB = dict( ((x[1], x[2]), x[0]) for x in listB )
joined = [ [ v, dictB.get(k), k[0], k[1] ] for k, v in dictA.iteritems() ]
```

Prehaps more knowledgeable pythonistas could comment on the pros and cons of these two different approaches (or maybe I'll post another question).

answered Nov 3 '14 at 21:57

Captain Whippet

374 1 14
