

join list of lists in python

Is there a short syntax for joining a list of lists into a single list(or iterator) in python?

For example I have a list as follows and I want to iterate over a,b and c.

```
x = [["a","b"], ["c"]]
```

The best I can come up with is as follows.

```
result = []
[ result.extend(el) for el in x]

for el in result:
    print el
```

python

asked Apr 4 '09 at 4:00



Kozyarchuk

4,901 7 28 36

Duplicate: stackoverflow.com/questions/120886/..., stackoverflow.com/questions/406121/... – S.Lott Apr 4 '09 at 11:52

[add a comment](#)

[start a bounty](#)

16 Answers

```
import itertools
a = [["a","b"], ["c"]]
print list(itertools.chain(*a))
```

answered Apr 4 '09 at 4:11



CTT

7,565 5 23 31

2 no need to list() it! for item in itertools.chain(*a): do something with item – hasenj Apr 4 '09 at 8:42

7 A bit of explanation would also be nice. docs.python.org/library/itertools.html#itertools.chain – hasenj Apr 4 '09 at 8:43

50 Slightly better: `list(itertools.chain.from_iterable(a))` – Neil G Jun 16 '11 at 22:45

36 `result = []; map(result.extend, a)` is ~30% faster than `itertools.chain`. But `chain.from_iterable` is a tiny bit faster than `map+extend`. [Python 2.7, x86_64] – temoto Jun 20 '11 at 2:23

3 This explains what's happening with the `*a`: stackoverflow.com/questions/5239856/foggy-on-asterisk-in-python (it sends the elements of `a` as arguments to `chain`, like removing the outer `[` and `]`). – Evgeni Sergeev Jan 9 at 6:00

[add a comment](#) | [show 1 more comment](#)

A performance comparison:

```
import itertools
import timeit
big_list = [[]*1000 for i in range(1000)]
timeit.repeat(lambda: list(itertools.chain.from_iterable(big_list)),
              number=100)
timeit.repeat(lambda: list(itertools.chain(*big_list)), number=100)
timeit.repeat(lambda: (lambda b: map(b.extend, big_list))([]), number=100)
timeit.repeat(lambda: [el for list_ in big_list for el in list_], number=100)
[100*x for x in timeit.repeat(lambda: sum(big_list, []), number=1)]
```

Producing:

```
>>> import itertools
>>> import timeit
>>> big_list = [[0]*1000 for i in range(1000)]
>>> timeit.repeat(lambda: list(itertools.chain.from_iterable(big_list)),
number=100)
[3.016212113769325, 3.0148865239060227, 3.0126415732791028]
>>> timeit.repeat(lambda: list(itertools.chain(*big_list)), number=100)
[3.019953987082083, 3.528754223385439, 3.02181439266457]
>>> timeit.repeat(lambda: (lambda b: map(b.extend, big_list))([]),
number=100)
[1.812084445152557, 1.7702404451095965, 1.7722977998725362]
>>> timeit.repeat(lambda: [el for list_ in big_list for el in list_],
number=100)
[5.409658160700605, 5.477502077679354, 5.444318360412744]
>>> [100*x for x in timeit.repeat(lambda: sum(big_list, []), number=1)]
[399.27587954973444, 400.9240571138051, 403.7521153804846]
```

This is with Python 2.7.1 on Windows XP 32-bit, but @temoto in the comments above got `from_iterable` to be faster than `map+extend`, so it's quite platform and input dependent.

Stay away from `sum(big_list, [])`

answered Jan 10 at 0:48

 [Evgeni Sergeev](#)
1,732 ●3 ●17 ●38

[add a comment](#)

For one-level flatten, if you care about speed, this is faster than any of the previous answers under all


conditions I tried. (That is, if you need the result as a list. If you only need to iterate through it on the fly then the chain example is probably better.) It works by pre-allocating a list of the final size and copying the parts in by slice (which is a lower-level block copy than any of the iterator methods):

```
def join(a):
    """Joins a sequence of sequences into a single sequence. (One-level
    flattening.)
    E.g., join([(1,2,3), [4, 5], [6, (7, 8, 9), 10]]) = [1,2,3,4,5,6,
    (7,8,9),10]
    This is very efficient, especially when the subsequences are long.
    """
    n = sum([len(b) for b in a])
    l = [None]*n
    i = 0
    for b in a:
        j = i+len(b)
        l[i:j] = b
        i = j
    return l
```


Sorted times list with comments:

```
[(0.5391559600830078, 'flatten4b'), # join() above.
(0.5400412082672119, 'flatten4c'), # Same, with sum(len(b) for b in a)
(0.5419249534606934, 'flatten4a'), # Similar, using zip()
(0.7351131439208984, 'flatten1b'), # list(itertools.chain.from_iterable(a))
(0.7472689151763916, 'flatten1'), # list(itertools.chain(*a))
(1.5468521118164062, 'flatten3'), # [i for j in a for i in j]
(26.696547985076904, 'flatten2')] # sum(a, [])
```

edited Oct 5 '12 at 17:47

 [esmit](#)
425 ●3 ●12

answered Jan 29 '12 at 2:51

 [Brandyn](#)
134 ●8

Can you add timings to confirm that this is faster than the other methods presented? – [esmit](#) Jun 27 '12 at 22:46

Sorted times list with comments: [(0.5391559600830078, 'flatten4b'), # join() above. (0.5400412082672119, 'flatten4c'), # Same, with sum(len(b) for b in a) (0.5419249534606934, 'flatten4a'), # Similar, using zip() (0.7351131439208984, 'flatten1b'), # list(itertools.chain.from_iterable(a)) (0.7472689151763916, 'flatten1'), # list(itertools.chain(*a)) (1.5468521118164062, 'flatten3'), # [i for j in a for i in j] (26.696547985076904, 'flatten2')] # sum(a, []) – [Brandyn](#) Jun 28 '12 at 9:09

You skipped `map(result.extend, a)` – [Kos](#) Dec 11 '12 at 8:00

There's a benchmark [ideone.com/9q3mrp](#) – [Kos](#) Dec 11 '12 at 8:35

@Kos, you are right! I'm lame. I probably omitted it originally because it "obviously" has bad O() time due to multiple copies, but now that I add it to my test, in practice it looks like it is successfully using realloc() to avoid this, and so it is winning hands down under all conditions. I remain skeptical, though, that it might revert to horrible behavior in a real working environment with fragmented memory. In a simple test app like this, with a clean slate of memory, it is free to keep extending the array without moving it. Thoughts? –

Brandyn Dec 11 '12 at 20:10

add a comment

This works recursively for infinitely nested elements:

```
def iterFlatten(root):
    if isinstance(root, (list, tuple)):
        for element in root:
            for e in iterFlatten(element):
                yield e
    else:
        yield root
```

Result:

```
>>> b = [ ["a", ("b", "c")], "d"]
>>> list(iterFlatten(b))
['a', 'b', 'c', 'd']
```


edited Sep 6 '12 at 9:25



Adam Harte

6,542 ●2 ●24 ●68

answered Apr 4 '09 at 9:20



Georg Schölly

66.5k ●24 ●140 ●208

>>> a = [] >>> a.append(a) >>> b = iterFlatten(a) >>> next(b) RuntimeError: maximum recursion depth exceeded in __instancecheck__ :) – Darthfett Mar 22 '12 at 16:39

1 @Darthfett would you expect a meaningful result for flattening an "infinitely-nested list"? :-) – Kos Dec 11 '12 at 7:46

@Kos A version that checks for such cases (by using a stack/set to check for self-references in a list) could be preferable than simply continuing to flatten until reaching the recursion depth limit. This could bypass the issue by simply giving the value, instead of trying to flatten it. – Darthfett Dec 11 '12 at 22:28

add a comment

I had a similar problem when I had to create a dictionary that contained the elements of an array and their count. The answer is relevant because, I flatten a list of lists, get the elements I need and then do a group and count. I used Python's map function to produce a tuple of element and it's count and groupby over the array. Note that the groupby takes the array element itself as the keyfunc. As a relatively new Python coder, I find it to me more easier to comprehend, while being Pythonic as well.

Before I discuss the code, here is a sample of data I had to flatten first:


```
{ "_id" : ObjectId("4fe3a90783157d765d000011"), "status" : [ "opencalais" ],
  "content_length" : 688, "open_calais_extract" : { "entities" : [
    {"type" : "Person", "name" : "Iman Samdura", "rel_score" : 0.223 },
    {"type" : "Company", "name" : "Associated Press", "rel_score" : 0.321
  }},
  {"type" : "Country", "name" : "Indonesia", "rel_score" : 0.321 }, ...
]},
"title" : "Indonesia Police Arrest Bali Bomb Planner", "time" : "06:42 ET",
"filename" : "021121bn.01", "month" : "November", "utctime" : 1037836800,
"date" : "November 21, 2002", "news_type" : "bn", "day" : "21" }
```

It is a query result from Mongo. The code below flattens a collection of such lists.

```
def flatten_list(items):
    return sorted([entity['name'] for entity in [entities for sublist in
    [item['open_calais_extract']['entities'] for item in items]
    for entities in sublist])
```

First, I would extract all the "entities" collection, and then for each entities collection, iterate over the dictionary and extract the name attribute.

answered Jul 31 '12 at 21:11



Karthik Gomadam

13 ●3

add a comment

If you need a list, not a generator, use `list()`:

```
from itertools import chain
x = [["a","b"], ["c"]]
y = list(chain(x))
```

answered Mar 18 '12 at 18:50



[culebrón](#)

5,849 ●4 ●29 ●63

s/ `x` / `*x` / (or `chain.from_iterable(x)` preferably) – [Kos](#) Dec 12 '12 at 8:52

I do not understand what it does. `join` is supposed to have a separator. – [Val](#) Oct 4 '13 at 15:45

@Val `chain` makes a generator that will output 'a', 'b', 'c'. `list` converts it into a list. – [culebrón](#) Oct 4 '13 at 19:52

[add a comment](#)

Late to the party but ...

I'm new to python and come from a lisp background. This is what I came up with (check out the var names for lulz):

```
def flatten(lst):
    if lst:
        car,*cdr=lst
        if isinstance(car,(list,tuple)):
            if cdr: return flatten(car) + flatten(cdr)
            return flatten(car)
        if cdr: return [car] + flatten(cdr)
        return [car]
```

Seems to work. Test:

```
flatten((1,2,3,(4,5,6,(7,8,(((1,2)))))))
```

returns:

```
[1, 2, 3, 4, 5, 6, 7, 8, 1, 2]
```

answered Dec 21 '10 at 3:19



[Michael Puckett](#)

201 ●3 ●7

12 You come from a lisp background? I never would have guessed from the code... haha – [Tom](#) Nov 17 '11 at 16:20

Nice, been doing Python for some time now and I haven't seen var-arg tuple unpacking like you did with `car, *cdr`. (e-> probably because it's Python 3 and I'm still digging 2 for some reason :-)) – [Kos](#) Dec 11 '12 at 7:52

[add a comment](#)

If you meant using "join":

```
print '\n'.join(map(str, listOfLists or listOfTuples))
```

answered Nov 19 '10 at 7:56



[Shikhar Mall](#)

11 ●1

[add a comment](#)

```
l = []
map(l.extend, list_of_lists)
```

shortest!

answered Jul 24 '10 at 1:04



[nate c](#)

4,095 ●10 ●19

2 `sum(listoflists,[])` # shorter! – [recursive](#) Jul 24 '10 at 1:18

1 @recursive Shorter but different functionally = much worse performance-wise, see comments on other variants for explanation – [Kos](#) Dec 11 '12 at 7:59

This tiny snippet appears to be the fastest way around for non-recursive flatten. Needs more upvotes –

This tiny snippet appears to be the fastest way around for non-recursive flatten. Needs more upvotes. — [Kos](#) Dec 11 '12 at 8:34

[add a comment](#)

Or a recursive operation:

```
def flatten(input):
    ret = []
    if not isinstance(input, (list, tuple)):
        return [input]
    for i in input:
        if isinstance(i, (list, tuple)):
            ret.extend(flatten(i))
        else:
            ret.append(i)
    return ret
```

answered Jan 2 '10 at 16:42



21 ● 1

[add a comment](#)

If you're only going one level deep, a nested comprehension will also work:

```
>>> x = [["a","b"], ["c"]]
>>> [inner
...     for outer in x
...     for inner in outer]
['a', 'b', 'c']
```

On one line, that becomes:


```
>>> [j for i in x for j in i]
['a', 'b', 'c']
```


answered Apr 4 '09 at 8:21



[George V. Reilly](#)

5,897 ● 3 ● 20 ● 28

Very cool, so for the next depth-level it will become [i for ll in x for l in ll for i in l] - at this point it starts getting a bit lame for the reader, but nevertheless cool :) — [Cyrus](#) Apr 10 '12 at 2:41 

For three levels, it gets nasty: >>> x = [[["a", "b"], ["c"]], [["d"]]] >>> [k for i in x for j in i for k in j] ['a', 'b', 'c', 'd'] — [George V. Reilly](#) Apr 17 '12 at 23:02 

[add a comment](#)

There's always reduce (being deprecated to functools):

```
>>> x = [ [ 'a', 'b'], ['c'] ]
>>> for el in reduce(lambda a,b: a+b, x, []):
...     print el
...
__main__:1: DeprecationWarning: reduce() not supported in 3.x; use
functools.reduce()
a
b
c
```



```
>>> import functools
>>> for el in functools.reduce(lambda a,b: a+b, x, []):
...     print el
...
a
b
c
>>>
```

Unfortunately the plus operator for list concatenation can't be used as a function -- or fortunate, if you prefer lambdas to be ugly for improved visibility.

answered Apr 4 '09 at 4:16



Aaron

2,484 ●13 ●20

1 GAH, I cannot believe they are deprecating it to functools. Anyway, you don't need the extra empty list, this will work just fine: `reduce(lambda a,b: a+b, x)` – Benson Apr 4 '09 at 6:23

2 Versions of the operators are defined as functions in the operator module, which are faster and less ugly than the lambda: `"functools.reduce(operator.add, [[1,2,3],[4,5]],[])"`. Alternatively, just use `sum()` – Brian Apr 4 '09 at 9:40

Personally, I think the lambda way is quite pretty. :-) – Benson Apr 16 '09 at 1:58

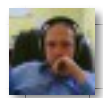
If you want to do a reduce, then reduce over `extend` not `add` to avoid spamming the memory with temporary lists. Wrap `extend` with a function that extends then returns the list itself. – Kos Dec 11 '12 at 7:58

[add a comment](#)

```
x = [["a","b"], ["c"]]
```

```
result = sum(x, [])
```

answered Apr 4 '09 at 4:15



CMS

338k ●91 ●626 ●694

24 $O(n^2)$ complexity yaaaaaay. – habnabit Sep 4 '10 at 5:57

1 @Aaron, explain for a noob python learner please: Is $O(n^2)$ good or bad in this case? ;-) – Aufwind Jul 11 '11 at 16:31

7 $O(n^2)$ here basically means that the time required for this function to execute is proportional to the square of the length of the inputs. So if you double the inputs, you quadruple the time required. This is a Bad Thing if you have large inputs, but for small ones it should be fine. But a faster method will be better. – andronikus Aug 29 '11 at 20:27

No, this is linear in the total number of items in all sublists combined, just like all other methods suggested on this page. – Julian Feb 26 at 23:38

1 @Julian: You are wrong. Just time it, or see stackoverflow.com/a/952952/279627. – Sven Marnach Feb 28 at 16:00

[add a comment](#) | [show 2 more comments](#)

Sadly, Python doesn't have a simple way to flatten lists. Try this:

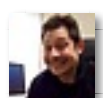
```
def flatten(some_list):
    for element in some_list:
        if type(element) in (tuple, list):
            for item in flatten(element):
                yield item
        else:
            yield element
```

Which will recursively flatten a list; you can then do

```
result = []
[ result.extend(el) for el in x]

for el in flatten(result):
    print el
```

answered Apr 4 '09 at 4:11



Don Werve

3,951 ●1 ●13 ●26

[add a comment](#)

What you're describing is known as **flattening** a list, and with this new knowledge you'll be able to find many solutions to this on Google (there is no built-in flatten method). Here is one of them, from <http://www.daniel-lemire.com/blog/archives/2006/05/10/flattening-lists-in-python/>:

```
def flatten(x):
    flat = True
    ans = []
    for i in x:
        if (i.__class__ is list):
            ans = flatten(i)
        else:
            ans.append(i)
    return ans
```

answered Apr 4 '09 at 4:08



[Jeremy Ruten](#)

65.6k ● 17 ● 110 ● 159

[add a comment](#)

This is known as flattening, and there are a LOT of implementations out there:

- [More on python flatten](#)
- [Python tricks](#)
- [Flattening lists in Python](#)

How about this, although it will only work for 1 level deep nesting:

```
>>> x = ["a","b"], ["c"]
>>> for el in sum(x, []):
...     print el
...
a
b
c
```

From those links, apparently the most complete-fast-elegant-etc implementation is the following:

```
def flatten(l, ltypes=(list, tuple)):
    ltype = type(l)
    l = list(l)
    i = 0
    while i < len(l):
        while isinstance(l[i], ltypes):
            if not l[i]:
                l.pop(i)
                i -= 1
                break
            else:
                l[i:i + 1] = l[i]
        i += 1
    return ltype(l)
```

edited Apr 4 '09 at 4:10

answered Apr 4 '09 at 4:04



[Paolo Bergantino](#)

212k ● 51 ● 403 ● 386

2 Ah, 'sum(L,l)' is shorthand for 'reduce(plus_operator, L, l)'. That's kinda cool. – [Aaron](#) Apr 4 '09 at 4:18

3 your "most complete-elegant-etc" is not "elegant" at all!! see the docs for `itertools.chain` to see true elegance! – [hasenj](#) Apr 4 '09 at 9:04

4 @hasenj: I believe he means best for arbitrary nested lists. `chain` assumes a consistent, one-deep list of lists (which is probably all the question needs), but `flatten` handles things like `[a,b,[c], [d,[e,f],[[g]]]]`. – [Brian](#) Apr 4 '09 at 9:44

[add a comment](#)

Not the answer you're looking for? Browse other questions tagged [python](#) or [ask your own question](#).