Elegant way to remove items from sequence in Python? [duplicate]

Asked 16 years, 4 months ago Modified 11 years, 9 months ago Viewed 48k times

57

This question already has answers here:

How to remove items from a list while iterating? (25 answers)

Closed 9 years ago.

When I am writing code in Python, I often need to remove items from a list or other sequence type based on some criteria. I haven't found a solution that is elegant and efficient, as removing items from a list you are currently iterating through is bad. For example, you can't do this:

```
for name in names:
    if name[-5:] == 'Smith':
        names.remove(name)
```

I usually end up doing something like this:

```
toremove = []
for name in names:
    if name[-5:] == 'Smith':
        toremove.append(name)
for name in toremove:
    names.remove(name)
del toremove
```

This is innefficient, fairly ugly and possibly buggy (how does it handle multiple 'John Smith' entries?). Does anyone have a more elegant solution, or at least a more efficient one?

How about one that works with dictionaries?

python optimization set series

Share

Improve this question

Follow

edited Oct 2, 2008 at 17:33

Andy Lester

93.5k ● 15 ● 104 ● 159

asked Aug 20, 2008 at 17:41

postfuturist

22.6k ● 11 ● 66 ● 85

Your code does remove multiple Smiths or did you edit it? – Geoffrey Jul 20, 2010 at 12:24

14 Answers

Sorted by: Highest score (default)

56

Two easy ways to accomplish just the filtering are:

1. Using `filter`:

```
names = filter(lambda name: name[-5:] != "Smith", names)
```

2. Using list comprehensions:

```
names = [name for name in names if name[-5:] != "Smith"]
```

Note that both cases keep the values for which the predicate function evaluates to `True`, so you have to reverse the logic (i.e. you say "keep the people who do not have the last name Smith" instead of "remove the people who have the last name Smith").

Edit Funny... two people individually posted both of the answers I suggested as I was posting mine.

Share

Improve this answer

Follow

edited Apr 10, 2012 at 5:32

Ray

192k ● 99 ● 227 ● 206

answered Aug 20, 2008 at 17:50

John

15.3k ● 12 ● 59 ● 57

12 `not name.endswith("Smith")` looks much nicer :-) – Jochen Ritzel Dec 7, 2009 at 4:13

5 sure, if you like readability or something. – John Dec 7, 2009 at 22:31

Can someone explain the `[-5:]` to me. What happens if you want to check the whole list? – Robert Johnstone Sep 14, 2011 at 15:45

3 @Sevenearths: The "[-5:]" takes the last five characters of the name, since we want to know whether the name ends with "Smith". As Jochen suggested, the expression "name[:-5]!='Smith'" might be written more readably as "not name.endswith('Smith')". – Edward Loper Nov 14, 2011 at 14:57

Don't forget to mention the performance increase by using `name.endswith("Smith")` instead of `[-5:]` – notbad.jpeg Jul 3, 2014 at 1:24

37

You can also iterate backwards over the list:

```
for name in reversed(names):
    if name[-5:] == 'Smith':
        names.remove(name)
```

This has the advantage that it does not create a new list (like `filter` or a list comprehension) and uses an iterator instead of a list copy (like `[:]`).

Note that although removing elements while iterating backwards is safe, inserting them is somewhat trickier.

Share Improve this answer Follow

answered Oct 8, 2008 at 1:24

Xavier Martinez-Hidalgo
618 ● 7 ● 11

This is a really innovative and Pythonic solution. I love it! – richo Dec 7, 2009 at 4:13

Does this work if there are duplicates in the list (that match the predicate)? – Jon-Eric Sep 4, 2012 at 16:58

@Jon-Eric: yes, it works. If there is a duplicate then the first one is removed, the list shrinks, and the `reversed()` yields the same `name` the second time. It is O(n**2) algorithm unlike the accepted answer that uses O(n) algorithm. – jfs May 23, 2014 at 21:18

29

The obvious answer is the one that John and a couple other people gave, namely:

```
>>> names = [name for name in names if name[-5:] != "Smith"]       # <-- slower
```

But that has the disadvantage that it creates a new list object, rather than reusing the original object. I did some profiling and experimentation, and the most efficient method I came up with is:

```
>>> names[:] = (name for name in names if name[-5:] != "Smith")    # <-- faster
```

Assigning to "names[:]" basically means "replace the contents of the names list with the following value". It's different from just assigning to names, in that it doesn't create a new list object. The right hand side of the assignment is a generator expression (note the use of parentheses rather than square brackets). This will cause Python to iterate across the list.

Some quick profiling suggests that this is about 30% faster than the list comprehension approach, and about 40% faster than the filter approach.

Caveat: while this solution is faster than the obvious solution, it is more obscure, and relies on more advanced Python techniques. If you do use it, I recommend accompanying it with a comment. It's probably only worth using in cases where you really care about the performance of this particular operation (which is pretty fast no

matter what). (In the case where I used this, I was doing A* beam search, and used this to remove search points from the search beam.)

Share Improve this answer Follow

answered Jan 9, 2011 at 14:41

Edward Loper

15.9k ● 7 ● 47 ● 52

2 Really interesting performance discovery. Could you share more about your profiling environment and evaluation methods? – Drake Guan Mar 2, 2012 at 1:39

I bet you could make it even faster by using `not name.endswith('Smith')` instead of creating a slice every iteration. Either way, this is a valuable piece of info that I probably never found if it weren't for your answer, thanks. – notbad.jpeg Jul 3, 2014 at 1:27

1 the `names[:]` suggestion was particularly helpful for using with `os.walk` to filter dirnames to traverse – wowest Jun 18, 2015 at 2:49

10

Using a list comprehension

```
list = [x for x in list if x[-5:] != "smith"]
```

Share Improve this answer Follow

answered Aug 20, 2008 at 17:49

Corey

14.3k ● 7 ● 39 ● 35

Does not really seem to be working for integers. temprevengelist = "0-12354-6876" temprevengelist = temprevengelist.split('-') list = [x for x in temprevengelist if x[-5:] != 6876] – Fahim Akhter Jan 28, 2010 at 7:48

@FahimAkhter: That's because you're comparing an integer to a string: in Python, `6876` (the int) and `"6876"` (the string) are two different values, and are not equal. Try replacing `x[-5:] != 6876` with either `x[-5:] != "6876"` or `int(x[-5:]) != 6876` – Edward Loper Apr 20, 2012 at 19:33

4

There are times when filtering (either using filter or a list comprehension) doesn't work. This happens when some other object is holding a reference to the list you're modifying and you need to modify the list in place.

```
for name in names[:]:
    if name[-5:] == 'Smith':
        names.remove(name)
```

The only difference from the original code is the use of `names[:]` instead of `names` in the for loop. That way the code iterates over a (shallow) copy of the list and the removals work as expected. Since the list copying is shallow, it's fairly quick.

Share Improve this answer Follow

answered Oct 5, 2008 at 11:48

elifiner

7,555 ● 8 ● 42 ● 48

3

filter would be awesome for this. Simple example:

```
names = ['mike', 'dave', 'jim']
filter(lambda x: x != 'mike', names)
['dave', 'jim']
```

Edit: Corey's list comprehension is awesome too.

Share Improve this answer Follow

answered Aug 20, 2008 at 17:49

mk.

26.2k ● 13 ● 39 ● 41

2

```
names = filter(lambda x: x[-5:] != "Smith", names);
```

Share Improve this answer Follow

answered Aug 20, 2008 at 17:48

pottedmeat

3,421 ● 1 ● 19 ● 9

2

Both solutions, *filter* and *comprehension* requires building a new list. I don't know enough of the Python internals to be sure, but I *think* that a more traditional (but less elegant) approach could be more efficient:

```
names = ['Jones', 'Vai', 'Smith', 'Perez']

item = 0
while item <> len(names):
    name = names [item]
    if name=='Smith':
        names.remove(name)
    else:
        item += 1

print names
```

Anyway, for short lists, I stick with either of the two solutions proposed earlier.

Share Improve this answer Follow

answered Aug 20, 2008 at 18:20

PabloG

26.6k ● 10 ● 48 ● 60

I think the names.remove(name) might be a O(n) operation, which would make this a O(n^2) algorithm. – postfuturist Oct 4, 2008 at 3:28

1 I would personally write my while expression as item < len(names), just in case I screwed up the logic inside the loop. (even though it doesn't look like you did) – Miquella Oct 8, 2008 at 0:31

It's probably more efficient to use del names[item] or names.pop(item) than names.remove(name). That's much less likely to be O(n), although I don't know the actual internals of how it works. – rjmunro Nov 5, 2008 at 13:11

2

To answer your question about working with dictionaries, you should note that Python 3.0 will include dict comprehensions:

```
>>> {i : chr(65+i) for i in range(4)}
```

In the mean time, you can do a quasi-dict comprehension this way:

```
>>> dict([(i, chr(65+i)) for i in range(4)])
```

Or as a more direct answer:

```
dict([(key, name) for key, name in some_dictionary.iteritems if name[-5:] !=
'Smith'])
```

Share

Improve this answer

Follow

edited Oct 8, 2008 at 0:17

answered Oct 7, 2008 at 14:33

Jason Baker

198k ● 138 ● 382 ● 520

you don't need to put `()` around generator expressions unless they aren't the only argument and `[]` makes the generator expression materialize a list which is what makes `dict([(k,v) for k,v in d.items()])` so much slower than `dict(((k,v) for k,v in d.items()))` – Dan D. Mar 4, 2011 at 10:40

If the list should be filtered in-place and the list size is quite big, then algorithms mentioned in the previous answers, which are based on list.remove(), may be

2

unsuitable, because their computational complexity is O(n^2). In this case you can use the following no-so pythonic function:

```
def filter_inplace(func, original_list):
  """ Filters the original_list in-place.

  Removes elements from the original_list for which func() returns False.

  Algrithm's computational complexity is O(N), where N is the size
  of the original_list.
  """

  # Compact the list in-place.
  new_list_size = 0
  for item in original_list:
    if func(item):
      original_list[new_list_size] = item
      new_list_size += 1

  # Remove trailing items from the list.
  tail_size = len(original_list) - new_list_size
  while tail_size:
    original_list.pop()
    tail_size -= 1


a = [1, 2, 3, 4, 5, 6, 7]

# Remove even numbers from a in-place.
filter_inplace(lambda x: x & 1, a)

# Prints [1, 3, 5, 7]
print a
```

Edit: Actually, the solution at https://stackoverflow.com/a/4639748/274937 is superior to mine solution. It is more pythonic and works faster. So, here is a new filter_inplace() implementation:

```
def filter_inplace(func, original_list):
  """ Filters the original_list inplace.

  Removes elements from the original_list for which function returns False.

  Algrithm's computational complexity is O(N), where N is the size
  of the original_list.
  """
  original_list[:] = [item for item in original_list if func(item)]
```

Share

Improve this answer

Follow

edited May 23, 2017 at 11:54

Community Bot

1 • 1

answered Apr 2, 2012 at 16:20

valyala

17.6k • 2 • 96 • 99

to remove trailing items: `del original_list[new_list_size:]` – jfs Mar 3, 2013 at 6:23

1

The filter and list comprehensions are ok for your example, but they have a couple of problems:

- They make a copy of your list and return the new one, and that will be inefficient when the original list is really big
- They can be really cumbersome when the criteria to pick items (in your case, if name[-5:] == 'Smith') is more complicated, or has several conditions.

Your original solution is actually more efficient for very big lists, even if we can agree it's uglier. But if you worry that you can have multiple 'John Smith', it can be fixed by deleting based on position and not on value:

```
names = ['Jones', 'Vai', 'Smith', 'Perez', 'Smith']

toremove = []
for pos, name in enumerate(names):
    if name[-5:] == 'Smith':
        toremove.append(pos)
for pos in sorted(toremove, reverse=True):
    del(names[pos])

print names
```

We can't pick a solution without considering the size of the list, but for big lists I would prefer your 2-pass solution instead of the filter or lists comprehensions

Share

Improve this answer

Follow

edited Oct 10, 2008 at 18:11

answered Oct 2, 2008 at 18:44

Ricardo Reyes

13.7k ● 4 ● 28 ● 19

This doesn't work properly if you have more than one 'Smith' entry, because the additional instances to remove have been shifted due to the removal of earlier instances. And for a similar reason, this algorithm causes an exception if a second 'Smith' entry is added to the end of the list. – Miquella Oct 8, 2008 at 0:37

@Miquella: you are right, my original post failed for multiple Smiths, I fixed it doing the delete in reverse order. Thanks. – Ricardo Reyes Oct 10, 2008 at 18:12

1

In the case of a set.

```
toRemove = set([])
for item in mySet:
    if item is unwelcome:
        toRemove.add(item)
mySets = mySet - toRemove
```

Share

Improve this answer

Follow

edited Dec 7, 2009 at 4:08

answered Dec 7, 2009 at 4:01

CashMonkey

11 ● 2

1

Here is my `filter_inplace` implementation that can be used to filter items from a list in-place, I came up with this on my own independently before finding this page. It is the same algorithm as what PabloG posted, just made more generic so you can use it to filter lists in place, it is also able to remove from the list based on the `comparisonFunc` if reversed is set `True`; a sort-of of reversed filter if you will.

```
def filter_inplace(conditionFunc, list, reversed=False):
    index = 0
    while index < len(list):
        item = list[index]

        shouldRemove = not conditionFunc(item)
        if reversed: shouldRemove = not shouldRemove

        if shouldRemove:
            list.remove(item)
        else:
            index += 1
```

Share Improve this answer Follow

answered Mar 15, 2013 at 14:12

Cory Gross

37.2k ● 17 ● 70 ● 80

-2

Well, this is clearly an issue with the data structure you are using. Use a hashtable for example. Some implementations support multiple entries per key, so one can either pop the newest element off, or remove all of them.

But this is, and what you're going to find the solution is, elegance through a different data structure, not algorithm. Maybe you can do better if it's sorted, or something, but iteration on a list is your only method here.

edit: one does realize he asked for 'efficiency'... all these suggested methods just iterate over the list, which is the same as what he suggested.

Share

Improve this answer

Follow

edited Aug 21, 2008 at 20:33

answered Aug 20, 2008 at 17:46

nlucaroni

47.9k ● 6 ● 66 ● 86

1 For some problems, switching to a different data structure isn't really an option -- in particular, if you don't know the filter condition until after the set of elements has been created. For example, if you're doing some sort of search, and want to prune your search space, you

generally won't know the appropriate cutoff condition for your pruning in advance.
– Edward Loper Jan 9, 2011 at 14:44