Removing duplicates in lists

Pretty much I need to w rite a program to check if a list has any duplicates and if it does it removes them and returns a new list v
w erent duplicated/removed. This is w hat I have but to be honest I do not know w hat to do.

```
def remove_duplicates():
    t = ['a', 'b', 'c', 'd']
    t2 = ['a', 'c', 'd']
    for t in t2:
        t.append(t.remove())
    return t
```

py thon algorithm list duplicates intersection

edited Feb 21 at 6:23
Raymond Hettinger
88.7k 16 179 26

7 Your description says you check "a list" for duplicates, but your code checks two lists.
– Brendan Long Nov 1 '11 at 0:48

23 Answers

The common approach to get a unique collection of items is to use a `set`. Sets are *unordered* collections of *distinct* objects. To create a set from any iterable, you can simply pass it to the built-in `set()` function. If you later need a real list again, you can similarly pass the set to the `list()` function.

The follow ing example should cover w hatever you are trying to do:

```
>>> t = [1, 2, 3, 1, 2, 5, 6, 7, 8]
>>> t
[1, 2, 3, 1, 2, 5, 6, 7, 8]
>>> list(set(t))
[1, 2, 3, 5, 6, 7, 8]
>>> s = [1, 2, 3]
>>> list(set(t) - set(s))
[8, 5, 6, 7]
```

As you can see from the example result, the original order is not maintained. As mentioned above, sets themselves are unordered collections, so the order is lost. When converting a set back to a list, an arbitrary order is created.

If order is important to you, then you will have to use a different mechanism. This question covers that topic in more detail.

edited Feb 3 '14 at 14:16 answered Nov 1 '11 at 0:49
poke
140k 27 229 291

91 It should be noted that this kills the original order. – Kos Jan 10 '13 at 9:00

20 It should be noted also that it doesn't work if you have dicts on the list. – fiatjaf Feb 23 '13 at 6:15

5 This isn't helpful if list order is important to you. – Jay Taylor May 21 '13 at 15:31

And most importantly, the content of the original list must be hashable. – Davide Feb 15 at 20:26

FWIW, the new (v2.7) Python way for removing duplicates from an iterable while keeping it in the original order is:

```
>>> from collections import OrderedDict
>>> list(OrderedDict.fromkeys('abracadabra'))
['a', 'b', 'r', 'c', 'd']
```

In Python 3.5, the OrderedDict has a C implementation. My timings show that this is now both the fastest and shortest of the various approaches.

In CPython 3.6, the regular dict in now both ordered and compact. For the moment, this is considered an implementation detail but will likely become a guaranteed feature in the future. That gives us a new fastest way of deduping while retaining order:

```
>>> list(dict.fromkeys('abracadabra'))
['a', 'b', 'r', 'c', 'd']
```

edited Mar 9 at 0:16 answered Nov 1 '11 at 0:53
Raymond Hettinger
88.7k 16 179 268

8 I think this is the only way to keep the items in order. – Herberth Amaral Oct 22 '12 at 20:23

11 @HerberthAmaral: That is very far from true, see How do you remove duplicates from a list in Python whilst preserving order? – Martijn Pieters♦ Aug 15 '13 at 14:24

2 @MartijnPieters Correcting: I think this is the only *simple* way to keep items in order. – Herberth Amaral Aug 15 '13 at 21:34

For this too, the content of the original list must be hashable – Davide Feb 15 at 20:28

It's a one-liner: `list(set(source_list))` will do the trick.

A `set` is something that can't possibly have duplicates.

edited Nov 1 '11 at 1:57 answered Nov 1 '11 at 0:49
9000
20.1k 4 34 64

```
>>> t = [1, 2, 3, 1, 2, 5, 6, 7, 8]
>>> t
```

```
[1, 2, 3, 1, 2, 5, 6, 7, 8]
>>> s = []
>>> for i in t:
       if i not in s:
          s.append(i)
>>> s
[1, 2, 3, 5, 6, 7, 8]
```

answered May 14 '13 at 12:39
Neeraj
529 4 2

15 Note that this method works in O(n^2) time and is thus very slow on large lists. – dotancohen Sep 3 '13 at 14:02

However this works fine for non-hashable content – Davide Feb 15 at 20:39

If you don't care about the order, just do this:

```
def remove_duplicates(l):
    return list(set(l))
```

A `set` is guaranteed to not have duplicates.

answered Nov 1 '11 at 0:49
Brendan Long
33.7k 8 96 137

To make a new list retaining the order of first elements of duplicates in `L`

```
newlist=[ii for n,ii in enumerate(L) if ii not in L[:n]]
```

for example `if L=[1, 2, 2, 3, 4, 2, 4, 3, 5]` then `newlist` will be `[1,2,3,4,5]`

This checks each new element has not appeared previously in the list before adding it. Also it does not need imports.

edited Aug 27 '14 at 23:14
FaCE
178 8

answered Jul 5 '14 at 3:39
Richard Fredlund
216 2 5

Another way of doing:

```
>>> seq = [1,2,3,'a', 'a', 1,2]
>> dict.fromkeys(seq).keys()
['a', 1, 2, 3]
```

edited Dec 3 '16 at 3:23
Aaron Hall ♦
73.5k 21 172 167

answered Jan 1 '14 at 15:39
James Sapam
6,190 3 20 38

A colleague have sent the accepted answer as part of his code to me for a codereview today. While I certainly admire the elegance of the answer in question, I am not happy with the performance. I have tried this solution (I use *set* to reduce lookup time)

```
def ordered_set(in_list):
    out_list = []
    added = set()
    for val in in_list:
        if not val in added:
            out_list.append(val)
            added.add(val)
    return out_list
```

To compare efficiency, I used a random sample of 100 integers - 62 w ere unique

```
from random import randint
x = [randint(0,100) for _ in xrange(100)]

In [131]: len(set(x))
Out[131]: 62
```

Here are the results of the measurements

```
In [129]: %timeit list(OrderedDict.fromkeys(x))
10000 loops, best of 3: 86.4 us per loop

In [130]: %timeit ordered_set(x)
100000 loops, best of 3: 15.1 us per loop
```

Well, w hat happens if set is removed from the solution?

```
def ordered_set(inlist):
    out_list = []
    for val in inlist:
        if not val in out_list:
            out_list.append(val)
    return out_list
```

The result is not as bad as w ith the *OrderedDict*, but still more than 3 times of the original solution

```
In [136]: %timeit ordered_set(x)
10000 loops, best of 3: 52.6 us per loop
```

answered Sep 17 '14 at 9:52
volcano
2,386 8 19

Nice using set quick lookup to speed up the looped comparison. If order does not matter list(set(x)) is still 6x faster than this – Joop Sep 17 '14 at 10:24

@Joop, that was my first question for my colleague - the order does matter; otherwise, it would have been trivial issue – volcano Sep 17 '14 at 11:00

I had a dict in my list, so I could not use the above approach. I got the error:

```
TypeError: unhashable type:
```

So if you care about **order** and/or some items are **unhashable**. Then you might find this useful:

```
def make_unique(original_list):
    unique_list = []
    [unique_list.append(obj) for obj in original_list if obj not in unique_list]
    return unique_list
```

Some may consider list comprehension w ith a side effect to not be a good solution. Here's an alternative:

```
def make_unique(original_list):
    unique_list = []
    map(lambda x: unique_list.append(x) if (x not in unique_list) else False,
original_list)
    return unique_list
```

edited Oct 27 '14 at 10:58
Dukeling
37.6k 9 44 86

answered Jun 6 '14 at 15:25
cchristelis
1,356 1 9 14

1 `map` with a side effect is even more misleading than a listcomp with a side effect. Also, `lambda x: unique_list.append(x)` is just a clunkier and slower way to pass `unique_list.append`. – abarnert Nov 8 '14 at 1:48

Try using sets:

```
import sets
t = sets.Set(['a', 'b', 'c', 'd'])
t1 = sets.Set(['a', 'b', 'c'])

print t | t1
print t - t1
```

answered Nov 1 '11 at 0:54
Charlie Martin
79.9k 15 139 222

Simple and easy:

```
myList = [1, 2, 3, 1, 2, 5, 6, 7, 8]
cleanlist = []
[cleanlist.append(x) for x in myList if x not in cleanlist]
```

Output:

```
>>> cleanlist
[1, 2, 3, 5, 6, 7, 8]
```

answered Apr 14 '15 at 23:33
Nima Soroush
3,267 1 28 34

quadratic complexity nonetheless - `in` is O(n) operation and your `cleanlist` will have at most `n` numbers => worst-case ~O(n^2) – jermenkoo Mar 23 '16 at 23:02

below code is simple for removing duplicate in list

```
def remove_duplicates(x):
    a = []
    for i in x:
        if i not in a:
            a.append(i)
    return a

print remove_duplicates([1,2,2,3,3,4])
```

it returns [1,2,3,4]

answered Aug 13 '15 at 21:54
vinay hegde
71 1 5

This is better than the accepted solution! :) Thanks – Buk Lau Sep 6 '15 at 11:24

If you don't care about order, then this takes significantly longer. `list(set(..))` (over 1 million passes) will beat this solution by about 10 whole seconds - whereas this approach takes about 12 seconds, `list(set(..))` only takes about 2 seconds! – dylnmc Sep 23 '16 at 18:35

You can use numpy function unique() (eventually using the function .tolist() if you don't w ant a numpy array)

```
import numpy as np
t=['a','a','b','b','b','c','c','c']
a=np.unique(t).tolist()
print a
>>>['a','b','c']
```

edited Jul 3 '14 at 16:41 answered Jul 3 '14 at 12:45
G M
1,573 1 14 30

This will convert the list to numpy array which is a mess and won't work for strings. – user227666 Jul 3 '14 at 12:48

1 @user227666 thanks for your review but that's not true it works even with string and you can add .tolist if you want to get a list... – G M Jul 3 '14 at 16:45

I think this is kinda like trying to kill a bee with a sledgehammer. Works, sure! But, importing a library for just this purpose might be a little overkill, no? – Debosmit Ray Oct 9 '16 at 9:11

@DebosmitRay it could be useful if you work in Data Science where usually you work with numpy and many times you need to work with numpy array. – G M Oct 10 '16 at 7:17

Now adays you might use Counter class:

```
>>> import collections
>>> c = collections.Counter([1, 2, 3, 4, 5, 6, 1, 1, 1, 1])
>>> c.keys()
dict_keys([1, 2, 3, 4, 5, 6])
```

answered Jun 18 '13 at 10:54
jb.
9,518 8 57 98

2 Why would someone do this instead of just using `dict` ? – Brendan Long Aug 13 '14 at 19:25

Here is an example, returning list w ithout repetiotions preserving order. Does not need any external imports.

```
def GetListWithoutRepetitions(loInput):
    # return list, consisting of elements of list/tuple loInput, without repetitions.
    # Example: GetListWithoutRepetitions([None,None,1,1,2,2,3,3,3])
    # Returns: [None, 1, 2, 3]

    if loInput==[]:
        return []

    loOutput = []

    if loInput[0] is None:
        oGroupElement=1
    else: # loInput[0]<>None
        oGroupElement=None

    for oElement in loInput:
        if oElement<>oGroupElement:
            loOutput.append(oElement)
            oGroupElement = oElement
    return loOutput
```

answered Jun 9 '14 at 10:33
Apogentus
1,899 14 21

This one cares about the order w ithout too much hassle (OrderdDict & others). Probably not the most Pythonic w ay, nor shortest w ay, but does the trick:

```
def remove_duplicates(list):
    ''' Removes duplicate items from a list '''
    singles_list = []
    for element in list:
        if element not in singles_list:
            singles_list.append(element)
    return singles_list
```

answered Sep 2 '14 at 11:37
cgf
1,233 2 18 32

Reduce variant w ith ordering preserve:

Assume that w e have list:

```
l = [5, 6, 6, 1, 1, 2, 2, 3, 4]
```

Reduce variant (unefficient):

```
>>> reduce(lambda r, v: v in r and r or r + [v], l, [])
[5, 6, 1, 2, 3, 4]
```

5 x faster but more sophisticated

```
>>> reduce(lambda r, v: v in r[1] and r or (r[0].append(v) or r[1].add(v)) or r, l, ([],
set()))[0]
[5, 6, 1, 2, 3, 4]
```

Explanation:

```
default = (list(), set())
# user list to keep order
# use set to make lookup faster

def reducer(result, item):
    if item not in result[1]:
        result[0].append(item)
        result[1].add(item)
    return result

reduce(reducer, l, default)[0]
```

edited Apr 27 '15 at 14:56

answered Apr 27 '15 at 14:42
Sergey M Nikitin
402 4 7

There are many other answ ers suggesting different w ays to do this, but they're all batch operations, and some of them throw aw ay the original order. That might be okay depending on w hat you need, but if you w ant to iterate over the values in the order of the first instance of each value, and you w ant to remove the duplicates on-the-fly versus all at once, you could use this generator:

```
def uniqify(iterable):
    seen = set()
    for item in iterable:
        if item not in seen:
            seen.add(item)
            yield item
```

This returns a generator/iterator, so you can use it anyw here that you can use an iterator.

```
for unique_item in uniqify([1, 2, 3, 4, 3, 2, 4, 5, 6, 7, 6, 8, 8]):
    print(unique_item, end=' ')

print()
```

Output:

```
1 2 3 4 5 6 7 8
```

If you do want a `list`, you can do this:

```
unique_list = list(uniqify([1, 2, 3, 4, 3, 2, 4, 5, 6, 7, 6, 8, 8]))

print(unique_list)
```

Output:

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

answered Aug 25 '15 at 23:51
Cyphase
6,067 11 25

`seen = set(iterable); for item in seen: yield item` is almost certainly faster. (I haven't tried this specific case, but that would be my guess.) – dylnmc Sep 23 '16 at 18:40

1 @dylnmc, that's a batch operation, and it also loses the ordering. My answer was specifically intended to be on-the-fly and in order of first occurrence. :) – Cyphase Oct 26 '16 at 4:42

All the order-preserving approaches I've seen here so far either use naive comparison (with O(n^2) time-complexity at best) or heavy-weight `OrderedDicts` / `set` + `list` combinations that are limited to hashable inputs. Here is a hash-independent O(nlogn) solution:

```
def filter_duplicates(lst):
    # Enumerate the list to restore order lately; reduce the sorted list; restore order
    def append_unique(acc, item):
        return acc if acc[-1][1] == item[1] else acc.append(item) or acc
    srt_enum = sorted(enumerate(lst), key=lambda (i, val): val)
    return [item[1] for item in sorted(reduce(append_unique, srt_enum, [srt_enum[0]]))]
```

edited Feb 3 '16 at 18:04
answered Jan 13 '16 at 19:12
Eli Korvigo
3,914 2 8 32

Yet, this solution requires orderable elements. I will use it uniquify my list of lists: it is a pain to `tuple()` lists and to hash them. | | | | - Generally speaking, the hash process takes a time proportional to the size of the whole data, while this solution takes a time O(nlog(n)), depending only on the length of the list. – loxaxs May 18 '16 at 20:40

I would really want to know, why someone has downvoted the answer. – Eli Korvigo Mar 3 at 12:11

To remove the duplicates, make it a SET and then again make it a LIST and print/use it. A set is guaranteed to have unique elements. For example :

```
a = [1,2,3,4,5,9,11,15]
b = [4,5,6,7,8]
c=a+b
print c
print list(set(c)) #one line for getting unique elements of c
```

The output will be as follows (checked in python 2.7)

```
[1, 2, 3, 4, 5, 9, 11, 15, 4, 5, 6, 7, 8]  #simple list addition with duplicates
[1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15] #duplicates removed!!
```

answered Aug 25 '15 at 8:38
krozaine
311 1 7

Check this if you want to remove duplicates (in-place edit rather than returning new list) without using inbuilt set, dict.keys, uniqify, counter

```
>>> t = [1, 2, 3, 1, 2, 5, 6, 7, 8]
>>> for i in t:
...     if i in t[t.index(i)+1:]:
...         t.remove(i)
...
>>> t
[3, 1, 2, 5, 6, 7, 8]
```

answered Nov 20 '15 at 15:20
user2404093
48 10

Use `enumerate()` to get the index faster: `for i, value in enumerate(t): if value in t[i + 1:]: t.remove(value)` – Martijn Pieters♦ Mar 9 '16 at 13:36

For completeness, and since this is a very popular question, the toolz library offers a `unique` function:

```
>>> tuple(unique((1, 2, 3)))
(1, 2, 3)
>>> tuple(unique((1, 2, 1, 3)))
(1, 2, 3)
```

answered Mar 9 at 11:50
Björn Pollex
50k 14 130 217

`unique_everseen` [1] can remove all duplicates while preserving the order:

```
>>> from iteration_utilities import unique_everseen

>>> list(unique_everseen(['a', 'b', 'c', 'd'] + ['a', 'c', 'd']))
['a', 'b', 'c', 'd']
```

In case you want to avoid the overhead of the list addition operation you can use `itertools.chain` instead:

```
>>> from itertools import chain
>>> list(unique_everseen(chain(['a', 'b', 'c', 'd'], ['a', 'c', 'd'])))
['a', 'b', 'c', 'd']
```

[1] Disclosure: I'm the author of the `iteration_utilities` -library.

edited Mar 1 at 2:52 answered Nov 9 '16 at 1:56
MSeifert
21.4k 10 33 64

protected by Brad Larson♦ Jul 31 '14 at 16:07

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, reputation on this site (the association bonus does not count).