sign up log in tour help careers 2.0

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required. Take the 2-minute tour ×

# How to clone or copy a list in Python?


What are the options to clone or copy a list in Python?

Using `new_list = my_list` then modifies `new_list` every time `my_list` changes. Why is this?

python | list | clone

edited Aug 6 at 17:12
wnnmaw
**1,852** 4 25

asked Apr 10 '10 at 8:49
aF.
**16.1k** 13 63 122

**3** I am allways changing one list and it was giving me a problem while iteranting the list later on the code. So I made a deepcopy of it and problem solved :) – aF. Apr 10 '10 at 13:53

**2** Why are you "always changing one list"? Why not create a new list combining "change" and "clone" into a single, simpler operation? – S.Lott Apr 12 '10 at 13:26

**50** S.Lott, why would you want to question someone so unhelpfully when they're just asking a reasonable question? – murftown Dec 11 '11 at 6:54

**3** (Not me that made the question, but I would imagine) because, very often, it's only through asking questions about methods, and being forced to answer questions on why we chose those methods, that we realise we actually had made incorrect assumptions and should have been using different methods in the first place. – scubbo May 21 '13 at 13:17

add a comment

## 6 Answers

You have various possibilities:

- You can slice it:

  ```
  new_list = old_list[:]
  ```

  Alex Martelli's opinion (at least back in 2007) about this is, that *it is a weird syntax and it does not make sense to use it ever*. ;) (In his opinion, the next one is more readable).

- You can use the built in `list()` function:

  ```
  new_list = list(old_list)
  ```

- You can use generic `copy.copy()` :

  ```
  import copy
  new_list = copy.copy(old_list)
  ```

  This is a little slower than `list()` because it has to find out the datatype of `old_list` first.

- If the list contains objects and you want to copy them as well, use generic `copy.deepcopy()` :

  ```
  import copy
  new_list = copy.deepcopy(old_list)
  ```

  Obviously the slowest and most memory-needing method, but sometimes unavoidable.

**Example:**

```
import copy

class Foo(object):
    def __init__(self, val):
         self.val = val

    def __repr__(self):
        return str(self.val)

foo = Foo(1)

a = ['foo', foo]
b = a[:]
c = list(a)
d = copy.copy(a)
e = copy.deepcopy(a)

# edit orignal list and instance
a.append('baz')
foo.val = 5

print "original: %r\n slice: %r\n list(): %r\n copy: %r\n deepcopy: %r" \
       % (a, b, c, d, e)
```

Result:

```
original: ['foo', 5, 'baz']
slice: ['foo', 5]
list(): ['foo', 5]
copy: ['foo', 5]
deepcopy: ['foo', 1]
```

edited Apr 12 '10 at 16:08

answered Apr 10 '10 at 8:55

Felix Kling
**253k** 39 365 442

**4** +1 for 3 options, you can add new_list = list(new_list) as 4th – Anurag Uniyal Apr 10 '10 at 9:06

**1** Classes should inherit `object` rather than nothing so that you are using new-style classes, i.e. `class Foo(object):` – Mike Graham Apr 10 '10 at 14:29

**3** `+= ['baz']` seems like an odd way to write `.append('baz')` . – Mike Graham Apr 10 '10 at 14:30

**2** @Felix, of course it doesn't change the effect you see here, but every time there's a needlessly suboptimal snippets posted here, there is a chance readers might come along and mimic them. – Mike Graham Apr 12 '10 at 14:53

**1** @Mike Graham: Mmh thats true. Ok then, I provide more optimal code ;) – Felix Kling Apr 12 '10 at 16:05

show **6** more comments

CAREERS 2.0 by stackoverflow + Have projects on GitHub? Import them easily to your profile

Felix already provided an excellent answer, but I thought I'd do a speed comparison of the various methods:

1. 10.59 - `copy.deepcopy(old_list)`
2. 10.16 - pure python `Copy()` method copying classes with deepcopy
3. 1.488 - pure python `Copy()` method not copying classes (only dicts/lists/tuples)
4. 0.325 - `for item in old_list: new_list.append(item)`
5. 0.217 - `[i for i in old_list]` (a list comprehension)
6. 0.186 - `copy.copy(old_list)`
7. 0.075 - `list(old_list)`
8. 0.053 - `new_list = []; new_list.extend(old_list)`
9. 0.039 - `old_list[:]` (list slicing)

So the fastest is list slicing. But be aware that `copy.copy()` , `list[:]` and `list(list)` , unlike `copy.deepcopy()` and the python version don't copy any lists, dictionaries and class instances in the list, so if the originals change, they will change in the copied list too and vice versa.

(Here's the script if anyone's interested or wants to raise any issues:)

```
from copy import deepcopy

class old_class:
    def __init__(self):
        self.blah = 'blah'

class new_class(object):
    def __init__(self):
        self.blah = 'blah'

dignore = {str: None, unicode: None, int: None, type(None): None}

def Copy(obj, use_deepcopy=True):
    t = type(obj)

    if t in (list, tuple):
        if t == tuple:
            # Convert to a list if a tuple to
            # allow assigning to when copying
            is_tuple = True
            obj = list(obj)
        else:
            # Otherwise just do a quick slice copy
            obj = obj[:]
            is_tuple = False

        # Copy each item recursively
        for x in xrange(len(obj)):
            if type(obj[x]) in dignore:
                continue
            obj[x] = Copy(obj[x], use_deepcopy)

        if is_tuple:
            # Convert back into a tuple again
            obj = tuple(obj)
```

**EDIT**: Added new-style, old-style classes and dicts to the benchmarks, and made the python version much faster and added some more methods including list expressions and `extend()`.

edited Jun 6 '13 at 15:31

Andy Hayden
**40.7k** 10 57 104

answered Apr 10 '10 at 10:16

cryo
**4,923** 3 17 29

+1 I like it :) But I think `deepcopy` performs even worse if you have *real* (in a sense of not being a string) objects in the list... – Felix Kling Apr 10 '10 at 10:23

I've added new/old type classes/dicts, thanks for the feedback – cryo Apr 10 '10 at 11:05

+1: Surprising thing is that `new_list = []; new_list.extend(L)` is faster than `new_list = list(L)` . – J.F. Sebastian Apr 11 '10 at 20:46

@JFSebastian: Maybe because `list()` takes also other types besides lists. (I think any iteratable). Maybe it does some type checking... – Felix Kling Apr 11 '10 at 21:00

@Felix: `list.extend()` also accepts any iterable, so it shouldn't be type checking. – J.F. Sebastian Apr 11 '10 at 23:59

show **3** more comments

---

Use `thing[:]`

```
>>> a = [1,2]
>>> b = a[:]
>>> a += [3]
>>> a
[1, 2, 3]
>>> b
[1, 2]
>>>
```

answered Apr 10 '10 at 8:53

Paul Tarjan
**14.2k** 17 109 180

add a comment

---

```
new_list = list(old_list)
```

answered Apr 10 '10 at 9:03

user285176
**173** 5

add a comment

---

Python's idiom for doing this is `newList = oldList[:]`

answered Apr 10 '10 at 8:53

erisco
**6,219** 14 28

add a comment

---

I've been told that Python 3.3+ adds list.copy() method, which should be as fast as slicing:

```
newlist = old_list.copy()
```

answered Jul 23 '13 at 12:32

techtonik
**3,093** 21 39

add a comment

---

## protected by Ashwini Chaudhary Feb 13 at 13:08

Thank you for your interest in this question. Because it has attracted low-quality answers, posting an answer now requires 10 reputation on this site.

Would you like to answer one of these unanswered questions instead?

**Not the answer you're looking for? Browse other questions tagged** python list clone **or ask your own question.**