Using 'in' to match an attribute of Python objects in an array

Asked 16 years, 4 months ago Modified 13 years, 1 month ago Viewed 16k times

58

I don't remember whether I was dreaming or not but I seem to recall there being a function which allowed something like,

```
foo in iter_attr(array of python objects, attribute name)
```

I've looked over the docs but this kind of thing doesn't fall under any obvious listed headers

python arrays iteration

Share
Improve this question
Follow

edited Aug 23, 2008 at 16:11
James A. Rosen
65.2k ● 62 ● 184 ● 263

asked Aug 3, 2008 at 13:19
Brendan
19.3k ● 19 ● 89 ● 114

8 Answers

Sorted by: Highest score (default)

49

Using a list comprehension would build a temporary list, which could eat all your memory if the sequence being searched is large. Even if the sequence is not large, building the list means iterating over the whole of the sequence before `in` could start its search.

The temporary list can be avoiding by using a generator expression:

```
foo = 12
foo in (obj.id for obj in bar)
```

Now, as long as `obj.id == 12` near the start of `bar`, the search will be fast, even if `bar` is infinitely long.

As @Matt suggested, it's a good idea to use `hasattr` if any of the objects in `bar` can be missing an `id` attribute:

```
foo = 12
foo in (obj.id for obj in bar if hasattr(obj, 'id'))
```

Share Improve this answer Follow

answered Sep 11, 2008 at 22:42

Will Harris

21.7k ● 12 ● 66 ● 64

12

Are you looking to get a list of objects that have a certain attribute? If so, a [list comprehension]() is the right way to do this.

```
result = [obj for obj in listOfObjs if hasattr(obj, 'attributeName')]
```

Share Improve this answer Follow

answered Aug 3, 2008 at 15:59

Matt

5,923 ● 2 ● 30 ● 19

10

you could always write one yourself:

```
def iterattr(iterator, attributename):
    for obj in iterator:
        yield getattr(obj, attributename)
```

will work with anything that iterates, be it a tuple, list, or whatever.

I love python, it makes stuff like this very simple and no more of a hassle than neccessary, and in use stuff like this is hugely elegant.

Share Improve this answer Follow

answered Aug 27, 2008 at 20:13

dwestbrook

5,268 ● 3 ● 26 ● 20

7

No, you were not dreaming. Python has a pretty excellent list comprehension system that lets you manipulate lists pretty elegantly, and depending on exactly what you want to accomplish, this can be done a couple of ways. In essence, what you're doing is saying "For item in list if criteria.matches", and from that you can just iterate through the results or dump the results into a new list.

I'm going to crib an example from [Dive Into Python]() here, because it's pretty elegant and they're smarter than I am. Here they're getting a list of files in a directory, then filtering the list for all files that match a regular expression criteria.

>   files = os.listdir(path)
>   test = re.compile("test\.py$", re.IGNORECASE)
>   files = [f for f in files if test.search(f)]

You could do this without regular expressions, for your example, for anything where your expression at the end returns true for a match. There are other options like using the filter() function, but if I were going to choose, I'd go with this.

Eric Sipple

Share
Improve this answer
Follow

edited Nov 24, 2011 at 9:57
Shadow Wizard
66.3k ● 26 ● 146 ● 209

answered Aug 3, 2008 at 14:30
saalon
3,644 ● 4 ● 34 ● 40

6

The function you are thinking of is probably `operator.attrgettter`. For example, to get a list that contains the value of each object's "id" attribute:

```
import operator
ids = map(operator.attrgetter("id"), bar)
```

If you want to check whether the list contains an object with an id == 12, then a neat and efficient (i.e. doesn't iterate the whole list unnecessarily) way to do it is:

```
any(obj.id == 12 for obj in bar)
```

If you want to use 'in' with attrgetter, while still retaining lazy iteration of the list:

```
import operator,itertools
foo = 12
foo in itertools.imap(operator.attrgetter("id"), bar)
```

Share Improve this answer Follow

answered Feb 5, 2011 at 8:10
shang
24.8k ● 3 ● 59 ● 86

5

What I was thinking of can be achieved using list comprehensions, but I thought that there was a function that did this in a slightly neater way.

i.e. 'bar' is a list of objects, all of which have the attribute 'id'

The mythical functional way:

```
foo = 12
foo in iter_attr(bar, 'id')
```

The list comprehension way:

```
foo = 12
foo in [obj.id for obj in bar]
```

In retrospect the list comprehension way is pretty neat anyway.

Share Improve this answer Follow

answered Aug 3, 2008 at 16:13
Brendan
19.3k ● 19 ● 89 ● 114

3

If you plan on searching anything of remotely decent size, your best bet is going to be to use a dictionary or a set. Otherwise, you basically have to iterate through every element of the iterator until you get to the one you want.

If this isn't necessarily performance sensitive code, then the list comprehension way should work. But note that it is fairly inefficient because it goes over every element of the iterator and then goes BACK over it again until it finds what it wants.

Remember, python has one of the most efficient hashing algorithms around. Use it to your advantage.

Share Improve this answer Follow

answered Aug 27, 2008 at 20:30
Jason Baker
198k ● 138 ● 382 ● 520

0

I think:

```
#!/bin/python
bar in dict(Foo)
```

Is what you are thinking of. When trying to see if a certain key exists within a dictionary in python (python's version of a hash table) there are two ways to check. First is the `has_key()` method attached to the dictionary and second is the example given above. It will return a boolean value.

That should answer your question.

And now a little off topic to tie this in to the *list comprehension* answer previously given (for a bit more clarity). *List Comprehensions* construct a list from a basic *for loop* with modifiers. As an example (to clarify slightly), a way to use the `in dict` language construct in a *list comprehension*:

Say you have a two dimensional dictionary **`foo`** and you only want the second dimension dictionaries which contain the key **`bar`**. A relatively straightforward way to do so would be to use a *list comprehension* with a conditional as follows:

```
#!/bin/python
baz = dict([(key, value) for key, value in foo if bar in value])
```

Note the **`if bar in value`** at the end of the statement**, this is a modifying clause which tells the *list comprehension* to only keep those key-value pairs which meet the conditional.** In this case **`baz`** is a new dictionary which contains only the dictionaries from foo which contain bar (Hopefully I didn't miss anything in that code example... you may have to take a look at the list comprehension documentation found in docs.python.org tutorials and at secnetix.de, both sites are good references if you have questions in the future.).

Share Improve this answer Follow

answered Aug 3, 2008 at 15:47

akdom
33.1k ● 27 ● 74 ● 79

Who in the world has their Python interpreter installed in `/bin`? – Willow Falzone Aug 18, 2020 at 16:28

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.