Are there builtin iterators in Python?

Asked 1 year, 6 months ago Active 19 days ago Viewed 205 times

0

We have builtin iterables like lists, tuples, and dictionaries to name a few. We can also create our own iterable objects by implementing an `__iter__` method in the class. We can also do that iterator objects by implementing an `__iter__` and a `__next__` method, but are there builtin iterators like there are builtin iterables?

python iterator built-in

edited Aug 20 '18 at 9:37
Graham Dumpleton
51.7k ● 6 ● 95 ● 116

asked Aug 20 '18 at 9:21
multigoodverse
4,660 ● 8 ● 40 ● 81

3 Answers

3

The following `builtin`s *return* iterators in Python 3: `enumerate()`, `filter()`, `iter()` (of course), `map()`, `reversed()` and `zip()`.

In Python there are also a lot of native python methods that *return* iterators, for example checkout the `itertools` module (the hint is in the name!).

However, to pedantically answer your question, no there are not `builtins` that *are* iterators (I can't think of a good use case for this), but as tobias_k says `list()` and others are not iterables either and merely *return* then.

Testing that iterator *classes* (not objects) exist in `builtins` (thanks to FHTMitchell):

```
import builtins
import collections.abc

def isiteratorclass(obj):
    if not isinstance(obj, type):
        return False
    return issubclass(obj, collections.abc.Iterator)


[key for key, value in vars(builtins).items() if isiteratorclass(value)]
# --> ['enumerate', 'filter', 'map', 'reversed', 'zip']
```

edited Aug 20 '18 at 21:35

answered Aug 20 '18 at 9:33
Chris_Rands
25k ● 8 ● 51 ● 89

@tobias_k Agreed, have deleted the code snippet, it wasn't that relevant anyway – Chris_Rands Aug 20 '18 at 9:46

1 @tobias_k If we're going full pedant, `list` and `tuple` are classes whose instances are iterables ;) – FHTMitchell Aug 20 '18 at 9:48

Actually, I don't think your `is_iterator` test was pointless, you just applied it to the wrong arguments, e.g. `is_iterator([1,2,3])` or `is_iterator(range(5))` are false, whereas `is_iterator(map(abs, [1,2,3]))` is true. – tobias_k Aug 20 '18 at 9:49

@FHTMitchell Yes I think the OP just used the wrong semantics and that has led us down a rather irrelevant diversion, I have edited the answer – Chris_Rands Aug 20 '18 at 9:50

0

File handles, for example, implement the iterator protocol:

```
f = open('file.txt')
next(f)
# first line
next(f)
# second line
```

answered Aug 20 '18 at 9:26

schwobaseggl

42.3k ● 3 ● 29 ● 51

0

This code lists each of the builtins that implement **iter**:

```
import builtins

for item in dir(builtins):
    class_ = getattr(builtins, item)
    if type(class_) is type:
        if hasattr(class_, '__iter__'):
            print(item)
```

Which prints the following iterable **classes**:

```
bytearray
bytes
dict
enumerate
filter
frozenset
list
map
range
reversed
set
str
tuple
zip
```

It would not be very useful to have an iterator in the standard library because iterators get exhausted. When you call next() on them enough times they eventually will not return anything.

Also you said that things like 'zip' were 'native methods'. That is not strictly true. 'zip' is a class (see above) and zip() creates an instance of the zip class.

It is also helpful to think of the terms 'function' and 'methods' as distinctly different. If your function is defined in a module (a .py file) then its called a function. If it is defined in a class, it's called a method.

Python is a bit confusing because things like int and str and type are actually classes, but we use them like built in functions.

It's pretty interesting to explore around in the builtins module. Just do:

```
import builtins
dir(builtins)
```

Then check the type of some of the things you find in there like:

```
>>> type(dir)
<class 'builtin_function_or_method'>

>>> type(zip)
<class 'type'>

>>> type(int)
<class 'type'>

>>> type(chr)
<class 'builtin_function_or_method'>

>>> type(type)
<class 'type'>
```

To see what they really are. You will gain much more insight to how python works just from doing this exercise.

answered Feb 25 at 18:16

TKK

1 ● 1