Function generators vs class generators in Python 3

Asked 6 years, 5 months ago Modified 6 years, 5 months ago Viewed 5k times

6

Why do function generators and class generators behave differently? I mean, with class generators I can use generator as many times as I want, but with function generators, I can only use it once? Why so?

```
def f_counter(low,high):
    counter=low
    while counter<=high:
        yield counter
        counter+=1

class CCounter(object):
    def __init__(self, low, high):
        self.low = low
        self.high = high
    def __iter__(self):
       counter = self.low
       while self.high >= counter:
            yield counter
            counter += 1

f_gen=f_counter(5,10)
for i in f_gen:
    print(i,end=' ')

print('\n')

for j in f_gen:
    print(j,end=' ')  #no output

print('\n')

c_gen=CCounter(5,10)
for i in c_gen:
    print(i,end=' ')

print('\n')

for j in c_gen:
    print(j,end=' ')
```

python python-3.x generator

2 Answers

Sorted by: Highest score (default)

12

Calling the `f_gen()` function produces an *iterator* (specifically, a *generator iterator*). Iterators can only ever be looped over once. Your class is **not** an iterator, it is instead an *iterable*, an object that can produce any number of *iterators*.

Your class produces a *new* generator iterator each time you use `for`, because `for` applies the `iter()` function on the object you pass in, which in turn calls `object.__iter__()`, which in your implementation returns a *new* generator iterator each time it is called.

In other words, you can make the class behave the same way by calling `iter(instance)` or `instance.__iter__()` before looping:

```
c_gen = CCounter(5,10)
c_gen_iterator = iter(c_gen)
for i in c_gen_iterator:
    # ...
```

You can also make the `CCounter()` into an *iterator* by returning `self` from `__iter__`, and adding an `object.__next__()` method (`object.next()` in Python 2):

```
class CCounter(object):
    def __init__(self, low, high):
        self.low = low
        self.high = high
    def __iter__(self):
       return self
    def __next__(self):
       result = self.low
       if result >= self.high:
           raise StopIteration()
       self.low += 1
       return result
```

Share Improve this answer Follow

edited Aug 10, 2016 at 11:40

answered Aug 10, 2016 at 11:32

Martijn Pieters ♦
1.0m 282 3947 3283

```
class CCounter(object):
    def __init__(self, low, high):
        self.low = low
        self.high = high
        self.counter = self.low
    def __iter__(self):
       return self
    def __next__(self):
       if self.counter > self.high:
           raise StopIteration()
       val = self.counter
       self.counter += 1
       return val
```

Share Improve this answer Follow

answered Aug 10, 2016 at 11:32

Blckknght
97.7k 11 115 165