SciComp with Py

Generators

Vladimir Kulyukin
Department of Computer Science
Utah State University

Outline

- From Iterators to Generators
- Generator Expressions

Introduction

- Generators are one of the coolest and most useful features of Py
- Two application domains where Py generators shine are data analysis and sysadmin; I am sure there are many others
- This lecture is not meant to be an exhaustive tutorial; if you want to become closely familiar with generators, pursue the references given at the end of this lecture; generators never cease to surprise and amaze me

My Discovery of Generators

- I was happy to use iterators until I got into large, GB/TB-size, data processing projects
- Generators initially caught my attention, because I am always looking for NoSQL solutions to various data processing problems; my favorite solutions combine functional programming, regular expressions, pipelines, persistent objects, processes, etc.
- I do not really know if Py generators are the best way to approach big data processing/analysis projects (lots of different views on this issue!) but they yield excellent programmatic insights

From Iterators to Generators

Iterables

Many Py objects are iterables. For example, file streams are iterables that can be used to iterate over files line by line.

```
for line in open('data.txt'):
    print line
```

Iterables

Many Py objects can be explicitly converted into iterables with **iter()** and then iterated over. Here is a list iterable.

```
items = ['a', 'b', 'c']
_list_iter = iter(items)
for i  in  _list_iter:  print i
```

Iterables

Many Py objects can be converted into iterables with **iter()** and iterated over. Here is a dictionary iterable.

```
d = {'key1' : 10, 'key2' : 20 }
_key_iter = iter(d)
for key in _key_iter: print key
```

Consumption of Iterables

Many standard Py functions consume iterables: **sum**(iter), **min**(iter), **max**(iter), **list**(iter), **tuple**(iter), **set**(iter), **dict**(iter), etc.

```
sum(xrange(101))
xr = xrange(101)
mid = (min(xr)+max(xr))/2.0
```

Generators

- Generators can be thought of as lazy iterators, except you do not have to worry about the iterator protocol
- Many iterables are containers that already contain all items, e.g., a dictionary or a list
- Generators never contain all items, which is what makes them efficient, instead they generate/produce one item at a time on demand

Example

This generator generates 1, 2, 3, 4. Anytime you see **yield**, you are dealing with a generator. This is as simple as it gets with generators.

```
def gen_1234():
    yield 1
    yield 2
    yield 3
    yield 4
```

Running Generators in Py2

Creating a generator doesn't start it running. It simply returns a generator object. You need to call **next()** to get it going.

```
>>> g = gen_1234()
>>> g
<generator object gen_1234 at 0x02BAC058>
>>> g.next()
1
>>> g.next()
2
>>> g.next()
3
>>> g.next()
4
>>> g.next()

Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    g.next()
StopIteration
```

Running Generators in Py3

You need to call **next(g)** to get values out of Py3 generator.

```
>>> g = gen_1234()
>>> next(g)
1
>>> next(g)
2
>>> next(g)
3
>>> next(g)
4
```

Running Generators In Py3

You can also use **send(None)** to get values out of Py3 generator.

```
>>> g = gen_1234()
>>> g.send(None)
1
>>> g.send(None)
2
>>> g.send(None)
3
>>> g.send(None)
4
```

Using Generators in Loops in Py2

Using generators in loops is the same as using iterators.

Py2 Tests

```
def test_01():
  gf = gen_1234()
  while True:
    try:
      print gf.next()
    except StopIteration:
      break
```

```
def test_02():
  for x in gen_1234():
    print x
```

Py2 Shell

```
>>> test_01()
1
2
3
4
>>> test_02()
1
2
3
4
```

Using Generators in Loops in Py3

Using generators in loops is the same as using iterators.

Py3 Tests

```
def test_01():
  gf = gen_1234()
  while True:
    try:
      print(next(gf))
    except StopIteration:
      break
```

```
def test_02():
  for x in gen_1234():
    print(x)
```

Py3 Shell

```
>>> test_01()
1
2
3
4
>>> test_02()
1
2
3
4
```

Generators w/ Arguments

You can parameterize your generators with input arguments.

Py2 Source

```
def gen_num_range(lower, upper):
    for i in xrange(lower, upper+1):
        yield i

for n in gen_num_range(1, 10):
    print n
```

Py3 source is the same except for print() and range()

Py2 Shell

```
>>>
1
2
3
4
5
6
7
8
9
10
```

Generators & Iterators vs. Iterables

- You can iterate over a generator only once
- You can iterate over an iterator only once
- You can iterate over an iterable as many times as you want: e.g., lists or dictionaries, once created, can be iterated multiple times

Generator Expressions

Generator Expressions

- List comprehension expressions build lists
- Generator expressions are similar except they produce generators w/o building any lists

This is a generator expression.

This is a list comprehension expression that builds a list.

```
>>> [i for i in xrange(11) if i % 2 == 0]
[0, 2, 4, 6, 8, 10]
>>> g = (i for i in xrange(11) if i % 2 == 0)
>>> list(g)
[0, 2, 4, 6, 8, 10]
```

Things to Remember about Generator Expressions

- Generator expressions do not construct iterables; they construct generators
- Generator expressions are used in iterations
- Once a generator has been iterated over, it cannot be restarted

Generator Expression Syntax

Generator Expression

```
(expression for x1 in y1 if cond1
            for x2 in y2 if cond2
            ...
            if cond)
```

Loop Equivalent

```
for x1 in y1:
    if cond1:
        for x2 in y2 :
            if cond2
                ...
                if cond: yield expression
```

Consumption of Generator Expressions

Outside parentheses can be dropped when calling functions that consume generator expressions.

```
>>> sum(n for n in xrange(11) if n % 2 == 0)
30
```

```
>>> sum(n*n for n in xrange(11) if n % 2 != 0)
165
```

References & Resources

If you want to know more about generators, you can start at the link below and then follow the links on that URL.

https://wiki.python.org/moin/Generators