Python Iterators

Stéphane Vialette

LIGM, Université Paris-Est Marne-la-Vallée

October 19, 2011

Introducing iterators

Description

- Python supports a concept of iteration over containers.
- This is implemented using two distinct methods; these are used to allow user-defined classes to support iteration.

`container.__iter__()`

Return an iterator object. If a container supports different types of iteration, additional methods can be provided to specifically request iterators for those iteration types. (An example of an object supporting multiple forms of iteration would be a tree structure which supports both breadth-first and depth-first traversal.)

Introducing iterators

`iterator.__iter__()`

Return the iterator object itself. This is required to allow both containers and iterators to be used with the `for` and `in` statements.

`iterator.next()`

Return the next item from the container. If there are no further items, raise the `StopIteration` exception.

Fibonacci

Example

```
class Fibonacci(object):
    def __init__(self):
        self.fn2 = 1 # "f_{n-2}"
        self.fn1 = 1 # "f_{n-1}"
    def next(self):
        # next() is the heart of any iterator
        # te the use of the following tuple to not only save lines of
        # code but also to insure that only the old values of self.fn1 and
        # self.fn2 are used in assigning the new values
        (self.fn1,self.fn2,oldfn2) = (self.fn1+self.fn2,self.fn1,self.fn2)
        return oldfn2
    def __iter__(self):
        return self

fibonacci = Fibonacci()
for i in fibonacci: # print 1 1 2 3 5 8 13 21
    print i,
    if i > 20:
        break
```

The iter function

Example

```
In [1]: iter
Out[1]: <built-in function iter>

In [2]: it = iter(range(2))

In [3]: it
Out[3]: <listiterator object at 0x84513ac>

In [4]: it.next()
Out[4]: 0

In [5]: it.next()
Out[5]: 1

In [6]: it.next()
---------------------------------------------------------------------------
StopIteration Traceback (most recent call last)
.../Python/Cours/iterator/src/<ipython console> in <module>()
StopIteration:
```

The iter function

Example

```
# You know understand what is happening in this innocent loop
#
# for i in range(10):
#     print i

it = iter(range(10))
while True:
    try:
        i = it.next()
        print i
    except:
        raise StopIteration
```

Tree traversal

Example

```
class Tree(object):
    def __init__(self, label, left_tree = None, right_tree = None):
        self.label = label
        self.left_tree = left_tree
        self.right_tree = right_tree

tree = Tree('a',
            Tree('b'),
            Tree('c',
                 Tree('d'),
                 Tree('e',
                      Tree('f'),
                      Tree('g'))))
```

Tree traversal

Example

```
class Tree(object):
    ...
    class TreeIterator(object):
        def __init__(self, tree):
            self.stack = [tree]
        def __iter__(self):
            return self
        def next(self):
            if not self.stack:
                raise StopIteration
            tree = self.stack.pop(-1)
            if tree.right_tree is not None:
                self.stack.append(tree.right_tree)
            if tree.left_tree is not None:
                self.stack.append(tree.left_tree)
            return tree
    def __iter__(self):
        return Tree.TreeIterator(self)
```

Tree traversal

Example

```
class TreeIterator(object):
    def __init__(self, tree):
        self.stack = [tree]
    def __iter__(self):
        return self

class InOrder(TreeIterator):
    def next(self):
        if not self.stack:
            raise StopIteration
        tree = self.stack.pop(-1)
        if tree.right_tree() is not None:
            self.__stack.append(tree.right_tree)
        if tree.left_tree is not None:
            self.__stack.append(tree.left_tree)
        return tree
```

Substring iterator

Base class

```
class SubstringIterator(object):
    "Iterator over all non-empty substrings of a string"
    def __init__(self, seq):
        self.seq = seq
        self.indices = self.valid_indices()
    def valid_indices(self):
        return [(i, j)
                for i in range(len(self.seq))
                for j in range(1,len(self.seq)+1)
                if i < j]
    def __iter__(self):
        return self
    def next(self):
        try:
            i, j = self.indices.pop()
            return self.seq[i:j]
        except IndexError:
            raise StopIteration
```

Substring iterator

Testing

```
In [1]: from substringit import SubstringIterator

In [2]: for f in SubstringIterator("abcd"):
   ...:     print f,
   ...:
d cd c bcd bc b abcd abc ab a

In [3]: print " ".join([f for f in SubstringIterator("abcd")])
------> print(" ".join([f for f in SubstringIterator("abcd")]))
d cd c bcd bc b abcd abc ab a

In [4]: print " ".join([f for f in SubstringIterator("")])
------> print(" ".join([f for f in SubstringIterator("")]))


In [5]:
```

Substring iterator

Prefix and suffix

```
class PrefixIterator(SubstringIterator):
    "Iterator over all non-empty prefixes of a string"
    def valid_indices(self):
        return [(0, j) for j in range(len(self.seq)+1)]

class SuffixIterator(SubstringIterator):
    "Iterator over all non-empty suffixes of a string"
    def valid_indices(self):
        return [(i, len(self.seq)+1) for i in range(len(self.seq))]
```

Substring iterator

Prefix and suffix

```
In [1]: from substringit import *

In [2]: print " ".join([f for f in PrefixIterator("abcd")])
------> print(" ".join([f for f in PrefixIterator("abcd")]))
abcd abc ab a

In [3]: print " ".join([f for f in SuffixIterator("abcd")])
------> print(" ".join([f for f in SuffixIterator("abcd")]))
d cd bcd abcd

In [4]:
```

Distinct iterator

One (non-efficient) implementation

```
class DistinctIterator(object):
    """
    Iterate over all distinct values of an iterator
    """
    def __init__(self, it):
        self.it = it
        self.seen = []
    def __iter__(self):
        return self
    def next(self):
        while True:
            v = self.it.next()
            if v not in self.seen:
                self.seen.append(v)
                return v
```

Distinct iterator

Testing

```
In [1]: from substringit import *

In [2]: print " ".join([f for f in SubstringIterator("aaaaa")])
------> print(" ".join([f for f in SubstringIterator("aaaaa")]))
a aa a aaa aa a aaaa aaa aa a aaaaa aaaa aaa aa a

In [3]: print " ".join([f for f in DistinctIterator(SubstringIterator("aaaaa"))])
------> print(" ".join([f for f in DistinctIterator(SubstringIterator("aaaaa"))]))
a aa aaa aaaa aaaaa

In [4]: print " ".join([f for f in DistinctIterator(PrefixIterator("aaaaa"))])
------> print(" ".join([f for f in DistinctIterator(PrefixIterator("aaaaa"))]))
aaaaa aaaa aaa aa a

In [5]: print " ".join([f for f in DistinctIterator(SuffixIterator("aaaaa"))])
------> print(" ".join([f for f in DistinctIterator(SuffixIterator("aaaaa"))]))
a aa aaa aaaa aaaaa
```

Predicate iterator

One implementation

```
class PredicateIterator(object):
    """
    Iterate over all values of an iterator for which a predicate hold
    """
    def __init__(self, it, p):
        self.values = [v for v in it if p(v)]
    def __iter__(self):
        return iter(self.values)
```

Predicate iterator

Testing

```
In [1]: from substringit import *

In [2]: print " ".join([f for f in PredicateIterator(SubstringIterator("ababa"),
                                                    lambda _: True)])

a ba b aba ab a baba bab ba b ababa abab aba ab a

In [3]: print " ".join([f for f in PredicateIterator(SubstringIterator("ababa"),
                                                    lambda _: False)])

In [4]: print " ".join([f for f in PredicateIterator(SubstringIterator("ababa"),
                                                    lambda x: 'a' in x)])

a ba aba ab a baba bab ba ababa abab aba ab a

In [5]: print " ".join([f for f in PredicateIterator(SubstringIterator("ababa"),
                                                    lambda x: len(x) == 2)])
ba ab ba ab
```

Predicate iterator

Testing

```
In [1]: from substringit import *

In [2]: it = PredicateIterator(SubstringIterator("ababa"),
                               lambda x: 'a' in x)

In [3]: print " ".join([f for f in DistinctIterator(it)])

AttributeError Traceback (most recent call last)
.../Python/Cours/iterator/src/<ipython console> in <module>()
.../Cours/iterator/src/substringit_complete.pyc in next(self)
     34     def next(self):
     35         while True:
---> 36             v = self.it.next()
     37             if v not in self.seen:
     38                 self.seen.append(v)

AttributeError: 'PredicateIterator' object has no attribute 'next'
```

Subsequence iterator (step by step)

Right shift

```
In [1]: from operator import rshift

In [2]: ? rshift
Type:           builtin_function_or_method
Base Class:     <type 'builtin_function_or_method'>
String Form:    <built-in function rshift>
Namespace:      Interactive
Docstring:
    rshift(a, b) -- Same as a >> b.
In [3]: print ' '.join(["%d>>%d=%d" % (i, j, i>>j)
                       for i in range(5) for j in range(5) if i > j])

1>>0=1 2>>0=2 2>>1=1 3>>0=3 3>>1=1 3>>2=0 4>>0=4 4>>1=2 4>>2=1 4>>3=0

In [4]: 2>>-1
---------------------------------------------------------------------------
ValueError  Traceback (most recent call last)
.../Python/Cours/iterator/src/<ipython console> in <module>()
ValueError: negative shift count
```

Subsequence iterator (step by step)

Binary bitwise operation

```
In [1]: for j in range(5):
   ...:     for i in range(j+1, 5):
   ...:         print "%d>>%d & 1 = %d : " % (i, j, i>>j & 1),
   ...:         if (i>>j & 1):
   ...:             print "True"
   ...:         else:
   ...:             print "False"
   ...:
1>>0 & 1 = 1 :  True
2>>0 & 1 = 0 :  False
3>>0 & 1 = 1 :  True
4>>0 & 1 = 0 :  False
2>>1 & 1 = 1 :  True
3>>1 & 1 = 1 :  True
4>>1 & 1 = 0 :  False
3>>2 & 1 = 0 :  False
4>>2 & 1 = 1 :  True
4>>3 & 1 = 0 :  False
```

Subsequence iterator (step by step)

Sets

```
In [1]: l = range(5) + range(6)

In [2]: l
Out[2]: [0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 5]

In [3]: s = set(l)

In [4]: print s
------> print(s)
set([0, 1, 2, 3, 4, 5])

In [5]: s.add(7)

In [6]: s
Out[6]: set([0, 1, 2, 3, 4, 5, 7])

In [7]: s.remove(2)

In [8]: s
Out[8]: set([0, 1, 3, 4, 5, 7])
```

Subsequence iterator (step by step)

Sets

```
In [8]: s
Out[8]: set([0, 1, 3, 4, 5, 7])

In [9]: e = enumerate(s)

In [10]: e
Out[10]: <enumerate object at 0x84d0edc>

In [11]: for elt in e:
   ....:     print elt, ", "
   ....: print
(0, 0), (1, 1), (2, 3), (3, 4), (4, 5), (5, 7),
```

Subsequence iterator (step by step)

Sets

```
In [12]: # enumerate lists
         print ",".join([str(x) for x in enumerate(range(10))])
(0, 0),(1, 1),(2, 2),(3, 3),(4, 4),(5, 5),(6, 6),(7, 7),(8, 8),(9, 9)

In [18]: # enumerate dictionaries
         d = dict(a=1, b=3, c=5, d=7)

In [19]: d
Out[19]: {'a': 1, 'b': 3, 'c': 5, 'd': 7}

In [20]: print ",".join([str(x) for x in enumerate(d)])
(0, 'a'),(1, 'c'),(2, 'b'),(3, 'd')

In [21]:
```

Subsequence iterator

Implementation

```
class SubsequenceIterator(object):
    "Iterate over all non-empty subsequences of a string"
    def __init__(self, seq):
        self.seq = seq
        f = lambda x: [[y for j, y in enumerate(set(x))
                        if (i >> j) & 1]
                       for i in range(2**len(set(x)))]
        self.indices = f(range(len(seq)))
    def __iter__(self):
        return self
    def next(self):
        try:
            indices = self.indices.pop()
            if len(indices) == 0:
                return self.next()
            return ''.join([str(self.seq[i]) for i in indices])
        except IndexError:
            raise StopIteration
```