Python
Iterators

Stéphane Vialette

LIGM, Université Paris-Est Marne-la-Vallée

October 13, 2009

Outline

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.

Outline

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.next()
Out[3]: 0

In [4]: it.next()
Out[4]: 1

In [5]: it.next()
StopIteration Traceback (most recent call last)

In [6]:
```

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, l_tree = None, r_tree = None):
        self.__label = label
        self.__l_tree = l_tree
        self.__r_tree = r_tree
    def label(self):
        return self.__label
    def l_tree(self):
        return self.__l_tree
    def r_tree(self):
        return self.__r_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.r_tree() is not None:
                self.__stack.append(tree.r_tree())
            if tree.l_tree() is not None:
                self.__stack.append(tree.l_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.r_tree() is not None:
            self.__stack.append(tree.r_tree())
        if tree.l_tree() is not None:
            self.__stack.append(tree.l_tree())
        return tree
```