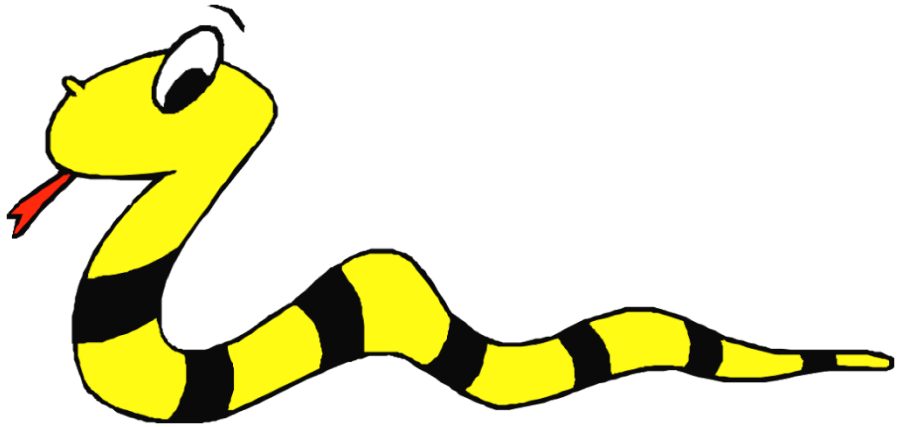
Python iterators and generators



Iterators and generators

- Python makes good use of iterators
- And has a special kind of generator function that is powerful and useful
- We'll look at what both are
- And why they are useful
- See Norman Matloff's excellent tutorial on python iterators and generators from which some of this material is borrowed

Files are iterators

```
>>> f = open("myfile.txt")
>>> for l in f.readlines(): print len(l)
9
21
35
43
>>> f = open("myfile.txt")
>>> for l in f: print len(l)
...
9
21
35
43
```

readlines() returns a list of the lines in file

A file is a iterator, producing new values as needed

Files are iterators

- Iterators are supported wherever you can iterate over collections in containers (e.g., lists, tuples, dictionaries)

```
>>> f = open("myfile.txt")
>>> map(len, f.readlines())
[9, 21, 35, 43]
>>> f = open("myfile.txt")
>>> map(len, f)
[9, 21, 35, 43]
>>>
```

Like sequences, but…

- Iterators are like sequences (lists, tuples), but…
- The entire sequence is not manifested
- Items produced one at a time when and as needed
- The sequence can be infinite (e.g., all positive integers)
- You can create your own iterators if you write a function to generate the next item

Example: fib.py

```
class fibnum:
    def __init__(self):
        self.fn2 = 1
        self.fn1 = 1

    def next(self): # next() is the heart of any iterator
          # use of the following tuple to not only save lines of
          # code but insures 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
```

next() used to generate successive values

Classes with an __iter__() method are iterators

http://cs.umbc.edu/courses/331/fall10/code/python/itgen/fib.py

Example: fib.py

```
>>> from fib import *
>>> f = fibnum()
>>> for i in f:
...   print i
...   if I > 100: break
1
1
2
3
…
144
>>>
```

http://cs.umbc.edu/courses/331/fall10/code/python/itgen/fib.py

Stopping an iterator

```
class fibnum20:
    def __init__(self):
        self.fn2 = 1 # "f_{n-2}"
        self.fn1 = 1 # "f_{n-1}"

    def next(self):
        (self.fn1,self.fn2,oldfn2) = (self.fn1+self.fn2,self.fn1,self.fn2)
        if oldfn2 > 20: raise StopIteration
        return oldfn2

    def __iter__(self):
        return self
```

Raise this error to tell consumer to stop

http://cs.umbc.edu/courses/331/fall10/code/python/itgen/fib.py

Stopping an iterator

```
>>> from fib import *
>>> for i in fibnum20(): print i
1
1
2
3
5
8
13
>>>
```

http://cs.umbc.edu/courses/331/fall10/code/python/itgen/fib.py

More tricks

- The list function materializes an iterator's values as a list

```
>>> list(fibnum20())
[1, 1, 2, 3, 5, 8, 13
```

- sum(), max(), min() know about iterators

```
>>> sum(fibnum20())
33
>>> max(fibnum20())
13
>>> min(fibnum20())
1
```

itertools

- The itertools library module has some useful tools for working with iterators
- islice() is like slice but works with streams produced by iterators

```
>>> from itertools import *
>>> list(islice(fibnum(), 6))
[1, 1, 2, 3, 5, 8]
>>> list(islice(fibnum(), 6, 10))
[13, 21, 34, 55]
```

- See also imap, ifilter, …

Python generators

- Python generators generate iterators
- They are more powerful and convenient
- Write a regular function and instead of calling return to produce a value, call yield instead
- When another value is needed, the generator function picks up where it left off
- Raise the StopIteration exception or call return when you are done

Generator example

```
def gy():
  x = 2
  y = 3
  yield x,y,x+y
  z = 12
  yield z/x
  yield z/y
  return
```

```
>>> from gen import *
>>> g = gy()
>>> g.next()
(2, 3, 5)
>>> g.next()
6
>>> g.next()
4
>>> g.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

http://cs.umbc.edu/courses/331/fall10/code/python/itgen/gen.py

Generator example: fib()

```
def fib( ):
    fn2 = 1
    fn1 = 1
    while True:
        (fn1,fn2,oldfn2) = (fn1+fn2,fn1,fn2)
        yield oldfn2
```

http://cs.umbc.edu/courses/331/fall10/code/python/itgen/gen.py

Generator example: getword()

```
def getword(fl):
  for line in fl:
    for word in line.split():
      yield word
  return
```

http://cs.umbc.edu/courses/331/fall10/code/python/itgen/gen.py

Remembers stack, too

```
def inorder(tree):
  if tree:
    for x in inorder(tree.left):
      yield x
      yield tree.dat
      for x in inorder(tree.right):
        yield x
```