Python para quem sabe Python

Turma 1, aula 2

Iteráveis & cia.

Relembrando: iteradores em Python

- Em Python iteradores são obtidos através da função embutida iter()
- A função built-in iter() obtém iteradores dos objetos de uma destas formas:
  - iter(obj) procura método obj.__iter__() e o invoca
  - Se obj não tem método __iter__:
    - iter() tenta acessar obj[0], e se isso funcionar, **cria um iterador**
    - obj.__getitem__(0) caracteriza o velho protocolo de sequências
- Se iter é invocada com 2 argumentos iter(f, s):
  - Neste caso iter() **cria um iterador** que invoca f() e devolve os valores produzidos, até que f() produza o valor s

http://docs.python.org/library/collections.html#collections-abstract-base-classes

Função iter com dois argumentos

- Se iter é invocada com 2 argumentos iter(f, s):
  - Objeto é invocado f() repetidamente até que o valor retornado seja igual a s (a sentinela)

```
with open('mydata.txt') as fp:
    for line in iter(fp.readline, ''):
        process_line(line)
```

http://docs.python.org/library/functions.html#iter

```
from random import randint

def dado():
    return randint(1,6)

# gera valores ate que um 6 seja sorteado
for r in iter(dado, 6): # 6 e' o "sentinela"
    print r
```

https://github.com/oturing/ppqsp/blob/master/iteraveis/demo_iter2.py

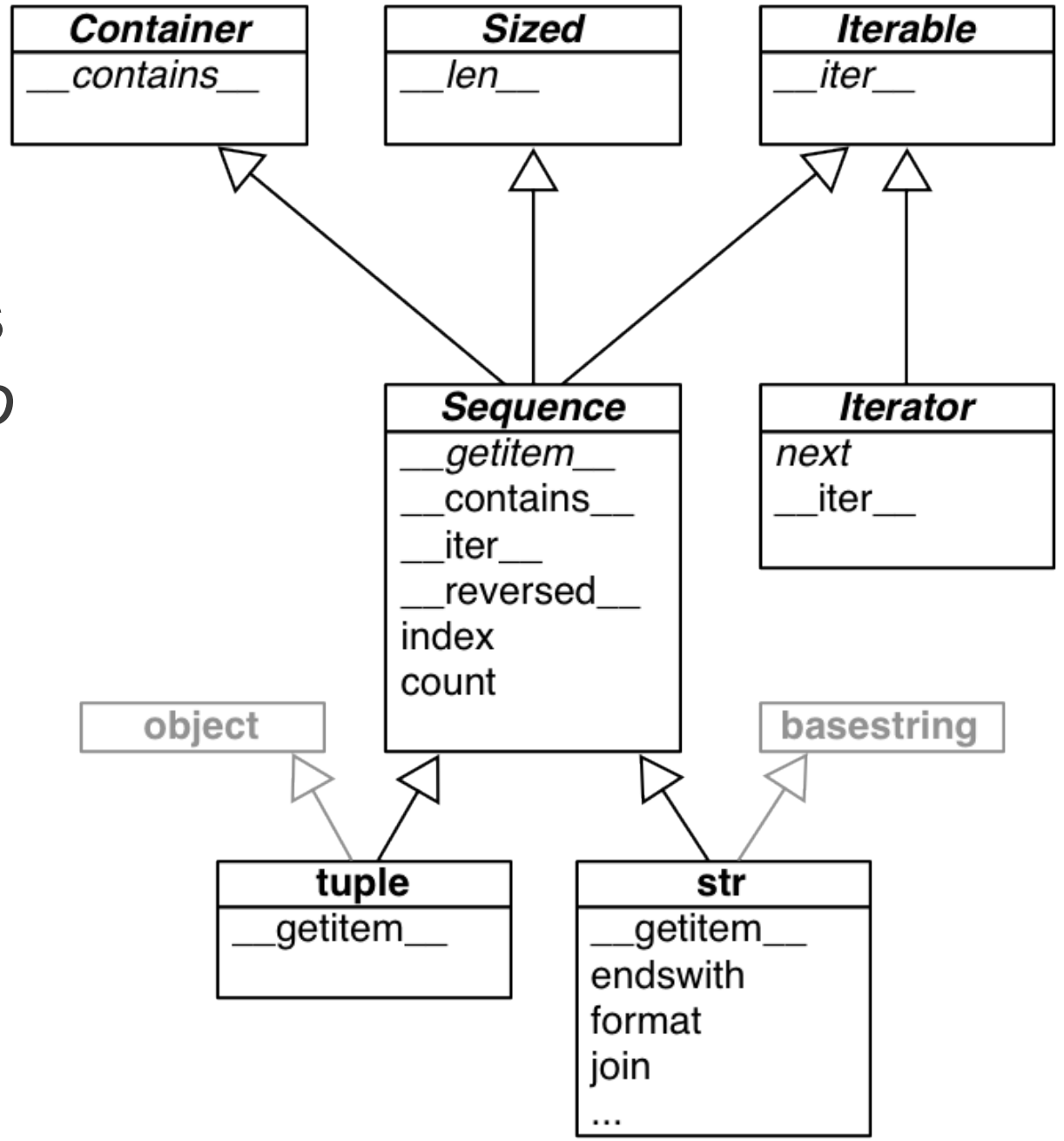
Tipos iteráveis: sequências

- Definição do protocolo (ou interface):
  - collections.Sequence
- Sequências imutáveis
  - str, unicode, tuple...
- Sequências mutáveis
  - list,array.array...

http://docs.python.org/library/collections.html#collections-abstract-base-classes

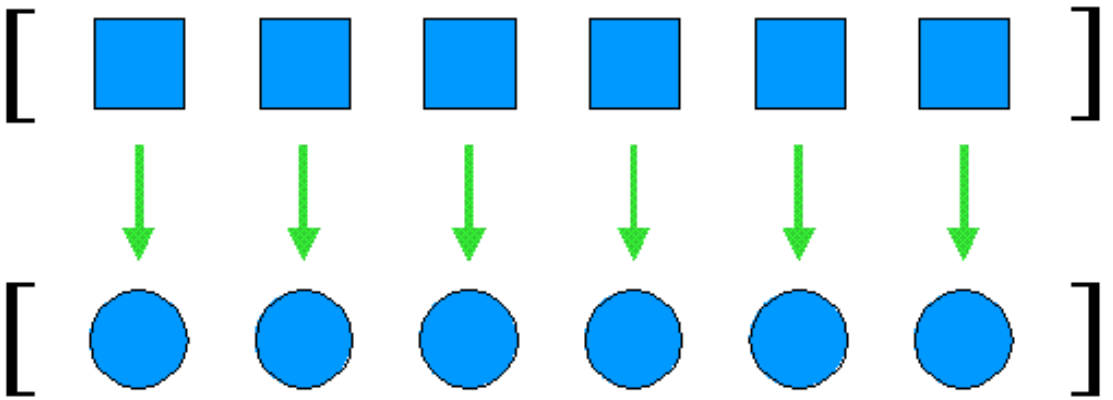
Sequence UML

- Classes e métodos abstratos em *itálico*
- Dois exemplos de subclasses de Sequence:
  - str
  - tuple



List comprehension

- Compreensão de lista ou abrangência de lista
- Exemplo: usar todos os elementos:
  - L2 = [n*10 for n in L]



List comprehension

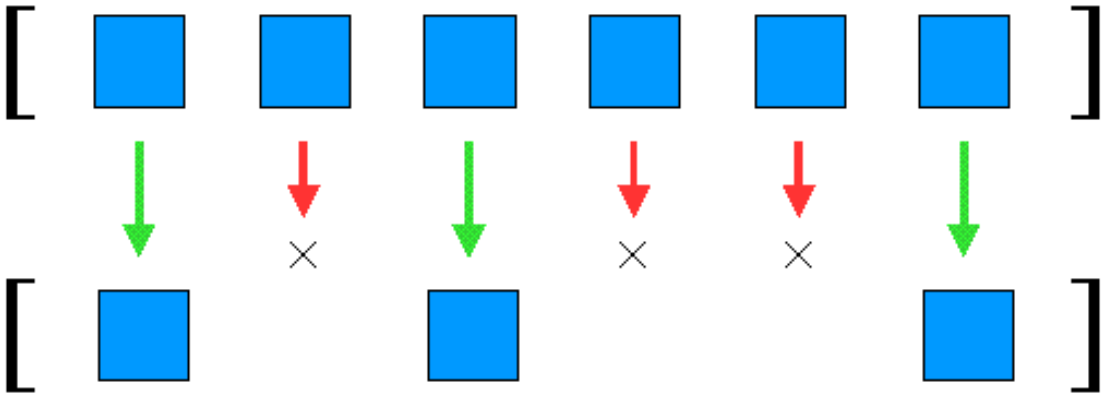
- Maior legibilidade, pela clareza da intenção
- Expressão **sempre** produz **nova lista**

```
>>> L = [8, 6.9, 6.4, 5]
>>> L2 = []
>>> for n in L:
...     L2.append(round(n,0))
...
>>> L2
[8.0, 7.0, 6.0, 5.0]
```

```
>>> L = [8, 6.9, 6.4, 5]
>>> L2 = [round(n,0) for n in L]
>>> L2
[8.0, 7.0, 6.0, 5.0]
```

List comprehension

- Filtrar alguns elementos:
  - L2 = [n for n in L if n > 0]



- Filtrar e processar
  - L2 = [n*10 for n in L if n > 0]

Iteráveis ansiosos x preguiçosos

- Exemplo built-in: sorted() x reversed()
- List comprehension x expressão geradora

```
>>> for i in [letra for letra in gera_letra()]:
...     print i
...
gerando 'A'...
gerando 'B'...
gerando 'C'...
A
B
C
```

```
>>> for i in (letra for letra in gera_letra()):
...     print i
...
gerando 'A'...
A
gerando 'B'...
B
gerando 'C'...
C
```

List comp. x expressão geradora

listcomp

genexp

```
from time import sleep, strftime

def demora(ts=1):
    agora = strftime('%H:%M:%S')
    print 'demorando...', agora
    sleep(1)
    return agora

raw_input('(tecle <ENTER> para rodar o teste LISTCOMP) ')

for t in [demora() for i in range(3)]:
    print t

raw_input('(tecle <ENTER> para rodar o teste GENEXP) ')

for t in (demora() for i in range(3)):
    print t
```

https://github.com/oturing/ppqsp/blob/master/iteraveis/demo_genexp.py

Funções geradoras

```
>>> def xxx():
...     yield 'X'
...     yield 'XX'
...     yield 'XXX'
...
>>> it = xxx()
>>> for i in it:
...     print i
...
X
XX
XXX
>>>
```

- O exemplo mais simples

```
>>> it = xxx()
>>> it
<generator object xxx at 0xb7840a2c>
>>> it.next()
'X'
>>> it.next()
'XX'
>>> it.next()
'XXX'
>>> it.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Função geradora

```
def gera_letra(ultima='C', verboso=True):
    cod = ord('A')
    while chr(cod) <= ultima:
        letra = chr(cod)
        if verboso:
            print 'gerando %r...' % letra
        yield letra
        cod += 1
```

```
>>> for i in [letra for letra in gera_letra()]:
...     print i
...
gerando 'A'...
gerando 'B'...
gerando 'C'...
A
B
C
```

```
>>> for i in (letra for letra in gera_letra()):
...     print i
...
gerando 'A'...
A
gerando 'B'...
B
gerando 'C'...
C
```

https://github.com/oturing/ppqsp/blob/master/iteraveis/genexp_x_listcomp.py

Funções geradoras

- Um exemplo prático: **isis2json.py**
- Objetivo dos geradores:
  - Desacoplar a lógica de leitura do arquivo da lógica de gravação
  - Sem iteradores, o laço principal do programa teria as duas lógicas entrelaçadas, tornando mais difícil a manutenção e principalmente a extensão para suportar mais formatos de entrada

https://github.com/bireme/isis2json/blob/master/isis2json.py

Funções embutidas que consomem ou produzem iteráveis

- Construtoras: list, dict, tuple, set, frozenset
  - Consome iterável, produz estrutura de dados
- Fábricas: range, zip, map, filter, sorted
  - Cria **nova lista** a partir de argumentos
- Geradoras: enumerate, reversed, xrange*
  - Cria gerador a partir de argumentos
- Redutoras: all, any, sum, min, max, len, reduce**
  - Reduz iterável a um único valor

* xrange não produz um gerador ou iterador de verdade, mas algo muito parecido

** reduce foi movida para o módulo functools no Python 3

Módulo itertools

- Iteradores (potencialmente) infinitos
  - count(), cycle(), repeat()
- Iteradores que combinam vários iteráveis
  - chain(), tee(), izip(), imap(), product(), compress()...
- Iteradores que selecionam ou agrupam itens:
  - compress(), dropwhile(), groupby(), ifilter(), islice()...
- Iteradores que produzem combinações
  - Ex: product(), permutations(), combinations()...

http://docs.python.org/library/itertools.html