Python para quem sabe Python

- Turma 1: 3ª/5ª 21h-23h (29/11; 1,6,8,13,15/12)
- Primeiras turmas
  - Agradeço a confiança, **mesmo!**
  - Satisfação garantida
- Estou sempre à disposição
  - E-mail: luciano@ramalho.org
  - Cel: +11-8432-0333 (use SMS)
  - Skype: LucianoRamalho

Funcionamento do curso

- Aula online ao vivo
- Rever gravação (opcional)
- Realizar tarefas até a próxima aula
- Discutir dúvidas e ajudar colegas no grupo
- Enxague e repita

Temas I

- sobrecarga de operadores (ou como fazer uma API Pythonica)
- iteráveis e iteradores (ou como percorrer qualquer coisa com um simples for)
- geradores e co-rotinas (ou outro jeito de organizar meus algoritmos)

Temas II

- acesso, criação e remoção dinânica de atributos (tudo sobre atributos básicos)
- propriedades e descritores (encapsulamento com atributos, ou como funcionam os modelos do Django)
- meta-programação, criação dinâmica de classes e funções (ou monkeypatching)
- metaclasses (ou como explodir sua mente criando classes turbinadas)

Temas III

- programação funcional (ou aquele outro paradigma que tá na moda)
- decoradores de funções (ou meta-programação com arte)
- programação assíncrona (ou como fazer muitas coisas ao mesmo tempo sem usar threads)
- gerenciadores de contexto (ou como usar o comando with)

1º exemplo: baralho polimórfico

- As cartas:

```
class Carta(object):
    def __init__(self, valor, naipe):
        self.valor = valor
        self.naipe = naipe

    def __repr__(self):
        return '<%s de %s>' % (self.valor, self.naipe)
```

```
>>> zape = Carta('4', 'paus')
>>> zape
<4 de paus>
```

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

1º exemplo: baralho polimórfico

- O baralho:

```
class Baralho(object):
    naipes = 'copas ouros espadas paus'.split()
    valores = 'A 2 3 4 5 6 7 8 9 10 J Q K'.split()

    def __init__(self):
        self.cartas = [Carta(v, n)
                       for n in self.naipes
                       for v in self.valores]

    def __getitem__(self, pos):
        return self.cartas[pos]

    def __len__(self):
        return len(self.cartas)
```

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

1º exemplo: baralho polimórfico

- O baralho funcionando:

```
>>> from baralho import Baralho
>>> b = Baralho()
>>> b[0]
<A de copas>
>>> b[:3]
[<A de copas>, <2 de copas>, <3 de copas>]
>>> b[-3:]
[<J de paus>, <Q de paus>, <K de paus>]
>>> for carta in b:
...     print carta
<A de copas>
<2 de copas>
<3 de copas>
```

O que é um iterável?

- Passo a pergunta para os universitários...

1º exemplo: baralho polimórfico

- O baralho funcionando:

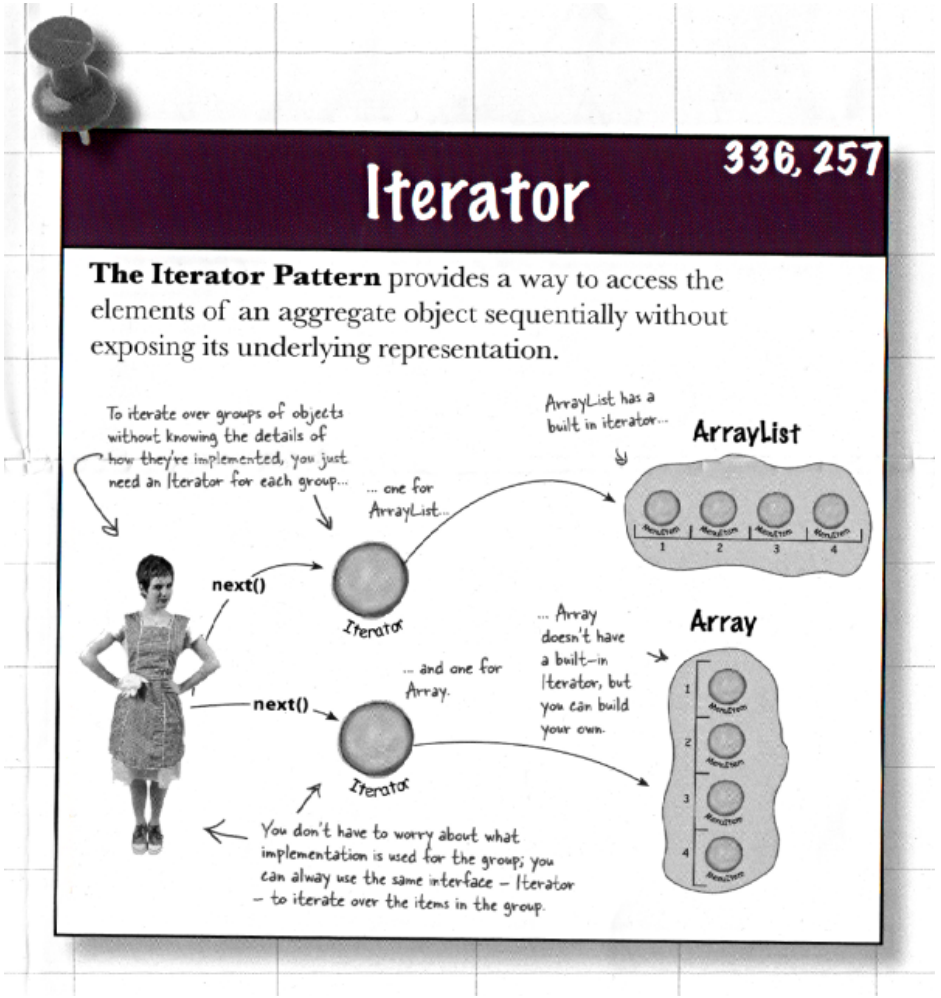
```
>>> from baralho import Baralho
>>> b = Baralho()
>>> b[0]
<A de copas>
>>> b[:3]
[<A de copas>, <2 de copas>, <3 de copas>]
>>> b[-3:]
[<J de paus>, <Q de paus>, <K de paus>]
>>> for carta in b:
...     print carta
<A de copas>
<2 de copas>
<3 de copas>
```

O que é um iterável?

- Um iterável é algo passível de ser iterado, ou seja, percorrido por uma iteração
- Um iterável em Python é um objeto a partir do qual a função iter(obj) consegue obter um iterador
- E o que é um iterador?

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

Iterador é um padrão de projeto



- Um iterador é um objeto que tem um método next(), através do qual acessamos os itens de uma coleção em sequência, sem precisarmos saber como a coleção é organizada internamente

Turing.com.br

Head First Design Patterns Poster

Como isso funciona em Python

- Um iterável em Python é um objeto a partir do qual a função iter() consegue obter um iterador
  - 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

O caso de uso mais comum de iter()

- Em Python moderno, as coleções implementam __iter__, e nesses casos a única coisa que o iter() faz é invocar obj.__iter__()
- O método __iter__ (dunder iter) tem a obrigação de devolver um **iterador**, que é um objeto que deve implementar os métodos:
  - next(self) # ou __next__(self) em Python 3
  - __iter__(self) # normalmente apenas devolve self
- Portanto, em Python iteradores são iteráveis!!!

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)
```

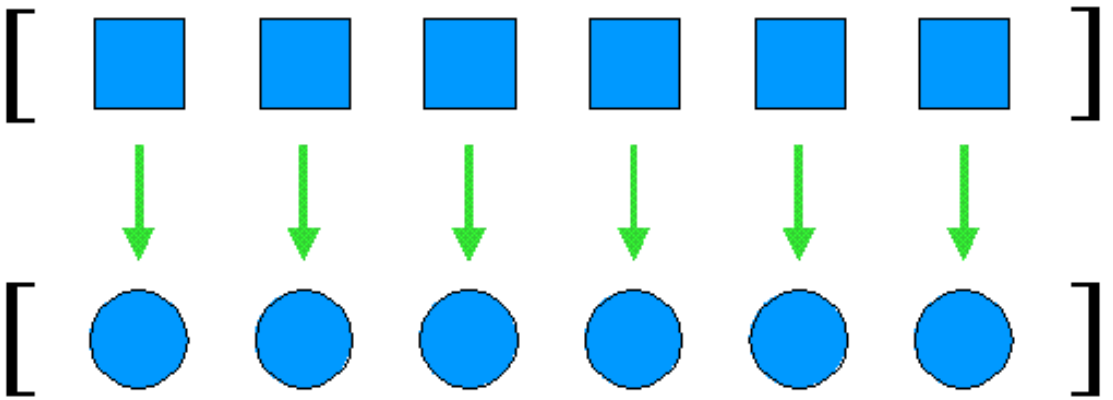
http://docs.python.org/library/functions.html#iter

```
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

List comprehension

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



List comprehension

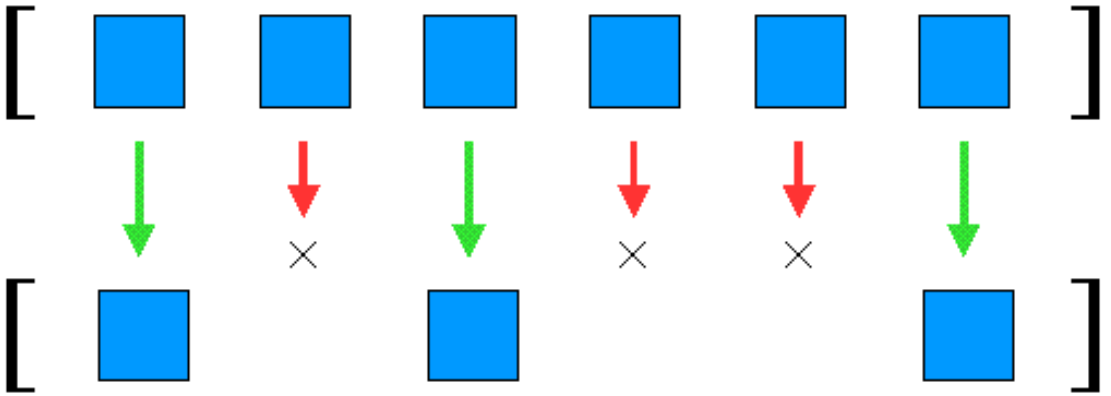
- Maior legibilidade, pela clareza da intenção
- Expressão **sempre** produz **nova lista**

```
>>> 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 busca_letra()]:
...     print i
...
buscando "A"
buscando "B"
buscando "C"
A
B
C
>>>
```

```
>>> for i in (letra for letra in busca_letra()):
...     print i
...
buscando "A"
A
buscando "B"
B
buscando "C"
C
>>>
```

Funções built-in (embutidas)

- 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

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

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