

Módulo 02 - Geradores

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



1. Geradores

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



Geradores

Gerador (ciência da computação)

🌐 15 línguas ▾

Artigo [Discussão](#)

[Ler](#) [Editar](#) [Ver histórico](#)

Origem: Wikipédia, a enciclopédia livre.

🔗 **Nota:** Para outros significados de *Generator*, veja [Generator](#).

Em [ciência da computação](#), um **gerador** (em [inglês](#), *generator*) é um [procedimento](#) especial que pode ser usado para controlar [iteradores](#) de [loops](#). Um gerador é muito similar para funções que retornam [arrays](#) (ou [vetores](#)), geradores podem ter parâmetros, que também podem ser chamados e geram uma sequência de valores. Entretanto, em vez de construir uma sequência que contenha todos os valores e os retornam de uma só vez, um gerador utiliza a palavra-chave *yield* para retornar os valores um de cada vez, que utiliza menos [memória](#) e permite o processamento de poucos valores rapidamente. Um gerador é uma função mas comporta-se como um [iterador](#).

As primeiras aparições de geradores foram na [CLU](#) em 1975, [\[1\]](#)  e atualmente são encontrados facilmente em [softwares](#) em [linguagem de programação Python](#) e [C#](#). (em CLU e C# generators são chamados de [iteradores](#)).

Geradores

- Uma função que representa uma coleção, podendo retornar múltiplas vezes **sem terminar** (mantendo o seu estado interno).
- Quando chamada, retorna o próximo elemento da coleção, ou sinaliza o seu término.

2. Geradores em Python

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



Python

```
for v in iter([1,2,3]):  
    print(v)
```

```
def f ():  
    yield 1  
    yield 2  
    yield 3  
  
for v in iter(f()):  
    print(v)
```

```
it = iter(f())  
print(it)  
print(next(it))  
print(next(it))  
print(next(it))  
print(next(it))
```

02-gen/01-python.py

- Iteração
 - for, next->next->next
- Gerador é um Iterador
 - objeto com next
 - next acorda gerador
 - yield “produz” um valor
- f vs f() vs next(f())

Retorna todos os naturais

```
def Nats ():  
    n = 0  
    while True:  
        yield(n)  
        n += 1  
  
for v in Nats():  
    print(v)
```

02-gen/02-nats.py

Exercício 2.2.1

- Crie um **iterador** em Python que retorna todos os números pares.
- Crie um **gerador** em Python que retorna todos os números pares.
- Faça uma discussão entre as diferenças fundamentais entre as duas abordagens.

```
for n in iter(Pares()):  
    print(n)
```

```
# saída  
1  
2  
3  
...
```


- lazy
- arvores
- infinito

1. Geradores em Python

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



Python

```
for v in [1,2,3]:  
    print(v)
```

```
for v in iter([1,2,3]):  
    print(v)
```

```
it = iter([1,2,3])  
print(it)  
print(next(it))  
print(next(it))  
print(next(it))  
print(next(it))
```

```
01-iter/01-python.py
```

- Iteração / Iteration
 - for, next->next->next
- Iterável / Iterable
 - [1,2,3], coleção
- Iterador / Iterator
 - it, objeto com next
- Função iter transforma um *iterável* em um *iterador*

Python - Iterator

- Um *Iterador* é um objeto que contém um número contável de valores.
- Um *Iterador* é um objeto que pode ser iterado, i.e., você pode percorrer todos os seus valores.
- Um *Iterador* é um objeto que implementa a interface com os métodos `__iter__` e `__next__`, e que pode gerar a exceção `StopIteration`.
- https://www.w3schools.com/python/python_iterators.asp

Python - Iterator

```
class MyIter:
    def __iter__(self):
        <...> # inicializa
    def __next__(self):
        <...> # retorna valor
        raise StopIteration # (ou)

ob = MyIter()
it = iter(ob)
next(it)
next(it)
<...>
```

```
class Sequencia:
    def __init__(self, v):
        self.max = v
    def __iter__(self):
        self.cur = 1
        return self
    def __next__(self):
        if self.cur > self.max:
            raise StopIteration
        v = self.cur
        self.cur += 1
        return v
```

```
for v in Sequencia(10):
    print(v)
```

```
ob = Sequencia(5)
it = iter(ob)
print(next(it))
```

Exercício 1.1.1

- Crie um iterador em Python que recebe uma string e retorna todos os caracteres da string, um por um.
 - Baseie-se na declaração a seguir...
 - Use o teste a seguir...

```
class Caracteres:  
    def __init__(self, str):  
        <...>  
    def __iter__(self):  
        <...>  
    def __next__(self):  
        <...>
```

```
for c in Caracteres("ola mundo"):  
    print(c)  
  
# saida  
o  
l  
...
```

01-iter/ex-1.1.1.lua
01-iter/ex-1.1.1.ceu

Exercício 1.1.2

- Considere um personagem que se move em um mapa nas 4 direções: “dir”, “esq”, “cima”, “baixo”.
- Crie um iterador em Python em que o personagem faz um movimento contínuo “em quadrado”, com 10 passos seguidos em cada direção.

```
class Quadrado:
    def __init__(self):
        <...>
    def __iter__(self):
        <...>
    def __next__(self):
        <...>
```

```
for dir in Quadrado():
    print(dir)

# saída
cima
cima
...
```

01-iter/ex-1.1.2.lua
01-iter/ex-1.1.2.ceu

2. Iteradores em Lua

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



Lua

```
for i,v in ipairs({10,20,30}) do
    print(v)
end
```

```
f,s,i0 = ipairs({1,2,3})
i1,v1 = f(s,i0)
print(i1,v1)
i2,v2 = f(s,i1)
print(i2,v2)
i3,v3 = f(s,i2)
print(i3,v3)
i4,v4 = f(s,i3)
print(i4,v4)
```

01-iter/03-lua.lua

- Iteração / Iteration
 - for, f(s,i)
- Iterável / Iterable
 - {1,2,3}
- Iterador / Iterator
 - f, s, i
- Função `ipairs` transforma um *vetor* em um *iterador*

Lua - Iterator

- O comando **for** genérico funciona usando funções, chamadas de *iteradores*.
- A cada iteração, a função iteradora é chamada para produzir um novo valor, parando quando esse novo valor é **nil**.
- O início do loop produz três valores: uma função iteradora, um estado, e um valor inicial para a variável de controle.
- A cada iteração, a função iteradora recebe dois valores: o estado e a variável de controle.
- A cada iteração, a função retorna, pelo menos, o próximo valor para a variável de controle.
- <http://www.lua.org/manual/5.2/pt/manual.html#3.3.5>

Python vs Lua

- Iteradores em Lua são *stateless*, pois não mantêm nenhum estado interno para controle da iteração.
- Iteradores em Python são *stateful*, pois dependem de um objeto para manter o estado interno da iteração.
- Iteradores em Lua podem ser *stateful*, por exemplo usando *closures*.

```
01-iter/ex-1.1.1-alt.lua
```

Exercício 1.2.1

- Crie um iterador em Lua que recebe uma árvore e retorne todas as folhas, da esquerda para a direita.
 - **Variante 1:** transforme a árvore em um vetor

```
a = {  
    'aaa',  
    {  
        'xxx',  
        'yyy',  
    },  
    'bbb'  
}
```

```
for f in Arvore(a) do  
    print(f)  
end  
  
-- saída  
aaa  
xxx  
yyy  
bbb
```

Exercício 1.2.2

- Crie um iterador em Python que recebe uma árvore e retorne todas as folhas, da esquerda para a direita.
 - **Variante 2:** use um pilha auxiliar como estado

```
a = [  
    "aaa",  
    [  
        "xxx",  
        "yyy"  
    ],  
    "bbb"  
]
```

```
for f in Arvore(a):  
    print(f)  
  
# saída  
aaa  
xxx  
yyy  
bbb
```

Módulo 02 - Geradores

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br



1. Geradores em Python

Tópicos em Linguagens

<https://github.com/fsantanna-uerj/LPX/>

Francisco Sant'Anna

francisco@ime.uerj.br

