

LINGUAGENS DE PROGRAMAÇÃO

Engenharia Informática

List comprehension

- Forma de compactar a criação de uma lista
 - *Ex: Criar uma nova lista com o incremento de cada elemento de outra lista*

```
lista = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
lista_mais_um = []  
for l in lista:  
    lista_mais_um.append(l+1)  
print (lista_mais_um)
```

#Ou

```
lista_mais_um = [ l + 1 for l in lista ]  
print (lista_mais_um)
```

List comprehension



```
lista = [ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
lista_pares = []
```

```
for l in lista:
```

```
    if l % 2 == 0:
```

```
        lista_pares.append(l)
```

```
print (lista_pares)
```

```
#Ou
```

```
lista_pares = [ l for l in lista if l % 2 == 0 ]
```

```
print (lista_pares)
```

Objeto lista é iterável



- O Objeto “lista” é iterável.
- Podia ser substituído por “range(10)”
 - *São abordagens parecidas mas o range é um gerador*
 - *Já lá vamos...*

Um objeto é iterável?

- O objeto pode ser representado através de uma sequência de elementos.
 - `print([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])`
`print(list(range(10)))`

O Objeto é:

- **Iterável** se implementa o método `__iter__(self)` e retorna um objeto iterador.
- **Iterador** se implementa `__next__` e este retorna o próximo elemento.
- **Iterável e Iterador** implementa ambas.

Porque existem iteráveis (e iteradores)

- Faz com que os objetos possam ser aplicados a for loops em vez de termos de indexar cada elemento.

```
lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
for i in lista:  
    print(i)
```

```
idx=0  
while idx < len(lista):  
    print(lista[idx])  
    idx += 1
```

Altamente ineficiente

Exemplo:

Iterar uma lista manualmente

- Podemos iterar manualmente utilizando o objeto iterador.

```
iterador = iter([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

while True:
    try:
        print( next(iterador) )
    except StopIteration:
        break
```

Implementa `__iter__(self)`
retorna iterador

Implementa `__next__(self)`

Alguns dentro de `__next__` existe um `raise StopIteration`

Geradores

- Imagine que temos de iterar todas as linhas de um ficheiro de 5GB.
 - *Vamos colocar tudo numa lista e iterar?*
 - *Pode ser!*
 - *Mas e se não houver memória?*
- Imagine que pretendemos iterar todas as pessoas no facebook. (se essa API fosse pública).
 - *Vamos carregá-las todas numa lista e iterar?*
 - *Memória? Talvez.*
 - *Carregar (download) e só depois iterar?*

- E se pudéssemos ir carregando e vizualizando?



Processa elemento
e passa ao seguinte.

E se pudéssemos ir gerando os elementos

Exemplo - Sem geradores

```
def obtem_pessoas_da_internet():  
    internet = ["João", "Francisco", "Manuel"]  
  
    lista_de_pessoas = []  
    for pessoa in internet:  
        print ("Download", pessoa)  
        lista_de_pessoas.append(pessoa)  
  
    return lista_de_pessoas  
  
pessoas = obtem_pessoas_da_internet()  
#pessoas é uma list que é iterável  
  
for p in pessoas: print(p)
```

```
Download João  
Download Francisco  
Download Manuel  
João  
Francisco  
Manuel
```

Exemplo - Com Funções Geradores

```
def obtem_pessoas_da_internet():  
    internet = ["João", "Francisco", "Manuel"]  
  
    for pessoa in internet:  
        print ("Download", pessoa)  
        yield pessoa  
  
    return
```

```
pessoas_g = obtem_pessoas_da_internet()  
#pessoas_g é um generator que é iterável  
  
for p in pessoas_g: print(p)
```

```
Download João  
João  
Download Francisco  
Francisco  
Download Manuel  
Manuel
```

Vantagem dos geradores:

- *Manter os recursos computacionais baixos.*
 - Quando a carga de memória aumenta bastante a performance do Python baixa.
- *Mantém um fluxo no tratamento dos elementos*
 - Permite aplicar filtros (if) quando aplicado em funções geradoras (ou expressões geradoras).
- *Mantém a mesma abordagem das listas*
 - Ambos são iteráveis.
 - O utilizador nem se apercebe que estão geradores a alimentar o for loop.

Limitações

- Os geradores são iteráveis e são o próprio iterador.
- Só podem ser utilizados uma vez (uma iteração)
 - *Tem de se criar um novo gerador(iterador) e percorrer de novo.*

Expressões geradoras

```
def obtem_pessoas_da_internet():  
    internet = ["João", "Francisco", "Manuel"]  
    return (x for x in internet)  
    #Retorna um gerador que itera sobre internet
```

```
pessoas = obtem_pessoas_da_internet()  
#pessoas é um gerador que é iterável e iterador
```

Pitfall

```
def obtem_pessoas_da_internet():  
    internet = ["João", "Francisco", "Manuel"]
```

```
    for pessoa in internet:  
        print ("Download", pessoa)  
        yield pessoa
```

```
pessoas_g = obtem_pessoas_da_internet()  
#pessoas_g é um gerador que é iterável
```

```
pessoas = list(pessoas_g)  
#pessoas é uma lista novamente
```

```
for p in pessoas: print(p)
```

```
Download João  
Download Francisco  
Download Manuel  
João  
Francisco  
Manuel
```

Fim

- Questões?