Comprehensions: ler, entender e utilizar

Alexandre Yukio Harano

Curitiba, 12 de Setembro de 2019 - Python Sul

alexandre@harano.net.br

https://alexandre.harano.net.br/

O que é comprehension?

Comprehension

Uma maneira compacta de processar todos os ou parte dos elementos de uma sequência e devolver (uma lista | um dicionário | um conjunto) com os resultados.

Adaptado de https://docs.python.org/3/glossary.html#term-list-comprehension





Comprehension

Extraído de https://docs.python.org/3/reference/expressions.html#list-displays



(1) https://github.com/ayharano/just-python/

Expressivo mecanismo para criação de uma nova instância de lista/dicionário/conjunto.



Expressivo mecanismo para criação de uma nova instância de lista/dicionário/conjunto..



Expressivo mecanismo para criação de uma nova instância de lista/dicionário/conjunto...



Expressivo mecanismo para criação de uma nova instância de lista/dicionário/conjunto... se usado cautelosamente.



```
lista = [
    EXPRESSÃO_DO_CONTEÚDO
    "for" ... "in" ...
    "if" ...
dicionário = {
    EXPRESSÃO DA CHAVE: EXPRESSÃO DO VALOR
    "for" ... "in" ...
    "if" ...
conjunto = {
    EXPRESSÃO_DO_VALOR
    "for" ... "in" ...
    "if" ...
```



Aloca os recursos necessários para **todos** os elementos que atendam os critérios.



Expressão Geradora

```
expressão_geradora = (
    EXPRESSÃO_DO_CONTEÚDO
    "for" ... "in" ...
    "if" ...
)
```



Expressão Geradora

Aloca os recursos necessários item a item conforme demanda.



Por que Expressão Geradora e não Generator Comprehension?

Resposta:

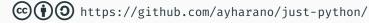
Comprehensions foram pensados inicialmente para aparentar com as estruturas de dados construídas (termo original: display) e geradores não possuem tal relação.

Mais detalhes em https://nedbatchelder.com/blog/201605/generator_comprehensions.html



Um Pequeno Desvio...

```
>>> lista = [x for x in range(0)]
>>> lista
Г٦
>>> bool(lista)
False
>>> expressão_geradora = (x for x in range(0))
>>> expressão geradora
<generator object <genexpr> at 0x7f4ee64f9bf8>
>>> bool(expressão geradora)
True
```



... E Como Contornar

```
>>> expressão_geradora = (x for x in range(0))
>>> expressão geradora
<generator object <genexpr> at 0x7f4ee64f9bf8>
>>> bool(expressão geradora)
True
>>> lista_da_expressão_geradora =
        list(expressão geradora)
>>> lista da expressão geradora
Г٦
>>> bool(lista da expressão geradora)
False
```



Iteradores Com Variáveis Dependentes

Iterações são lidas da expressão mais a esquerda para a direita.

Exemplo:

(0, 0)		
(0, 1)	(1, 1)	
(0, 2)	(1, 2)	(2, 2)



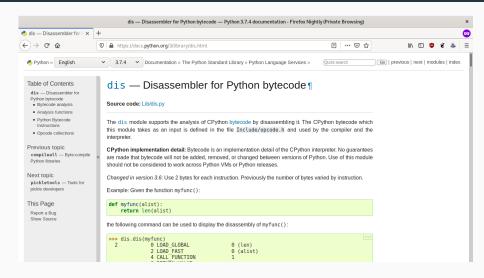
Asynchronous Comprehensions (PEP 530)

- Só pode ser utilizado dentro de funções com async def.
- Expressões podem ser calculadas a partir de
 - i Iteradores assíncronos (expr async for ...); ou
 - ii Uso de await em comprehensions regulares (await expr for ...).





dis





```
>>> def via nome(): return list()
>>> def via símbolo(): return []
>>> import dis
>>> dis.dis(via nome)
  1
              0 LOAD GLOBAL
                                          0 (list)
              2 CALL FUNCTION
              4 RETURN VALUE
>>> dis.dis(via símbolo)
              0 BUILD_LIST
                                          0
              2 RETURN VALUE
```



i) (1) https://github.com/ayharano/just-python/

Comparação

```
def usual():
    resultado = []
    for valor in range(100):
        if valor % 2:
            resultado.append(valor)
    return resultado
def list_comp():
    return [
        valor
        for valor in range(100)
        if valor % 2
def filter lambda():
    return list(
        filter(lambda valor: valor % 2, range(100))
```



Comparação (usual)

```
>>> dis.dis(usual)
  2
             0 BUILD LIST
                                        0
             2 STORE FAST
                                        0 (resultado)
  3
             4 SETUP LOOP
                                      34 (to 40)
             6 LOAD_GLOBAL
                                       0 (range)
             8 LOAD CONST
                                       1 (100)
            10 CALL FUNCTION
                                        1
            12 GET_ITER
           14 FOR_ITER
                                       22 (to 38)
       >>
            16 STORE FAST
                                       1 (valor)
            18 LOAD FAST
                                       1 (valor)
  4
            20 LOAD CONST
                                       2 (2)
            22 BINARY MODULO
            24 POP_JUMP_IF_FALSE
                                   14
  5
            26 LOAD_FAST
                                        0 (resultado)
            28 LOAD METHOD
                                       1 (append)
            30 LOAD FAST
                                        1 (valor)
            32 CALL METHOD
                                        1
            34 POP TOP
            36 JUMP ABSOLUTE
                                       14
            38 POP BLOCK
       >>
  6
       >>
            40 LOAD FAST
                                       0 (resultado)
            42 RETURN VALUE
        https://github.com/ayharano/just-python/
```

Comparação (list_comp)

```
>>> dis.dis(list comp)
              0 LOAD CONST
                                         1 (<code object <li>stcomp> at 0x7fddefe4ced0, file "
              2 LOAD_CONST
                                         2 ('list comp.<locals>.<listcomp>')
              4 MAKE FUNCTION
                                         0
                                         0 (range)
  4
             6 LOAD_GLOBAL
              8 LOAD CONST
                                         3 (100)
             10 CALL FUNCTION
             12 GET_ITER
             14 CALL FUNCTION
                                         1
             16 RETURN VALUE
Disassembly of <code object <li>stcomp> at 0x7fddefe4ced0, file "<stdin>", line 3>:
  3
              0 BUILD LIST
              2 LOAD FAST
                                         0 (.0)
              4 FOR ITER
                                        16 (to 22)
        >>
              6 STORE_FAST
                                         1 (valor)
  4
  5
             8 LOAD FAST
                                        1 (valor)
             10 LOAD CONST
                                         0 (2)
             12 BINARY MODULO
             14 POP JUMP IF FALSE
             16 LOAD FAST
                                         1 (valor)
             18 LIST_APPEND
             20 JUMP ABSOLUTE
             22 RETURN VALUE
```

Comparação (filter lambda)

```
>>> dis.dis(filter_lambda)
              0 LOAD GLOBAL
                                         0 (list)
  2
              2 LOAD_GLOBAL
                                         1 (filter)
  3
              4 LOAD CONST
                                         1 (<code object <lambda> at 0x7efd64690a50, file "<s
              6 LOAD_CONST
                                         2 ('filter_lambda.<locals>.<lambda>')
              8 MAKE FUNCTION
             10 LOAD GLOBAL
                                         2 (range)
             12 LOAD CONST
                                         3 (100)
             14 CALL FUNCTION
                                         1
             16 CALL FUNCTION
             18 CALL FUNCTION
             20 RETURN_VALUE
Disassembly of <code object <lambda> at 0x7efd64690a50, file "<stdin>", line 3>:
  3
              0 LOAD_FAST
                                         0 (valor)
                                         1 (2)
              2 LOAD CONST
              4 BINARY MODULO
              6 RETURN VALUE
```



```
import asyncio
async def intervalo progressivo(espera, até):
    for i in range(até):
        vield i
        await asyncio.sleep(espera)
async def async_set():
    return {
        valor
        async for valor in intervalo_progressivo(.001, 100)
        if valor % 2
```



```
>>> dis.dis(async_set)
10
              0 LOAD CONST
                                         1 (<code object <setcomp> at 0x7f4b340f0a50, file "a
                                         2 ('async set.<locals>.<setcomp>')
              2 LOAD CONST
              4 MAKE FUNCTION
                                         0
 12
             6 LOAD GLOBAL
                                         0 (intervalo progressivo)
              8 LOAD CONST
                                         3 (0.001)
             10 LOAD CONST
                                         4 (100)
             12 CALL FUNCTION
             14 GET AITER
             16 CALL_FUNCTION
                                         1
             18 GET AWAITABLE
             20 LOAD CONST
                                         0 (None)
             22 YIELD_FROM
             24 RETURN VALUE
```



. . .

```
Disassembly of <code object <setcomp> at 0x7f4b340f0a50, file "async set.py", line 10>:
10
             0 BUILD SET
             2 LOAD FAST
                                      0 (.0)
             4 SETUP EXCEPT
                                      12 (to 18)
        >>
             6 GET ANEXT
             8 LOAD CONST
                                        0 (None)
            10 YIELD FROM
 12
                                        1 (valor)
           12 STORE_FAST
            14 POP BLOCK
            16 JUMP FORWARD
                                     10 (to 28)
        >> 18 DUP TOP
            20 LOAD GLOBAL
                                      0 (StopAsvncIteration)
            22 COMPARE OP
                                      10 (exception match)
            24 POP JUMP IF TRUE
                                       42
            26 END_FINALLY
. . .
```



. . .

```
13
     >> 28 LOAD FAST
                                    1 (valor)
                                     1 (2)
          30 LOAD CONST
           32 BINARY MODULO
           34 POP_JUMP_IF_FALSE
           36 LOAD FAST
                                     1 (valor)
           38 SET ADD
          40 JUMP ABSOLUTE
      >> 42 POP TOP
          44 POP TOP
           46 POP TOP
           48 POP EXCEPT
           50 POP TOP
           52 RETURN VALUE
```



Não-Exemplos (Aprecie Com Moderação!)

```
resultado = [transformação_complexa(
              x, algum argumento=x+1)
             for x in iterável if predicado(x)
resultado = [
    (x, y)
    for x in range(10)
    for y in range(5)
    if x * y > 10
return ((x, y, z)
        for x in range(5)
        for v in range(5)
        if x != v
        for z in range(5)
        if v != z)
```



Pontos principais

- Comprehensions s\u00e30 utilizados para criar listas|dicion\u00e1rios|conjuntos.
- Pode melhorar a legibilidade de código principalmente para filtrar dados.
- Variáveis alocadas dentro da comprehensions só valem dentro do escopo local, ou seja, menos variáveis temporárias.
- Não recomendado usar com expressões que exijam tratamento (ex: Exceptions).
- Cautela com os recursos! Se souber de antemão que os dados extrapolam a memória, usar expressões geradoras.



Perguntas?

Alexandre Yukio Harano

alexandre@harano.net.br

https://alexandre.harano.net.br/