La philosophie de l'it?ration

September 22, 2017

```
In [1]: implementations = ['cpython', 'pypy', 'jython', 'ironpython']

        for implementation in implementations:
            print(implementation)
```

```
cpython
pypy
jython
ironpython
```

La philosophie de l'itération
kevin@formationspython.com

1 Un design pattern : iterator

Eviter de boucler à la main:

```
const char *relatedTopics[] = {
    "++",
    "jerome",
    "pas ma faute",
    NULL
};

for (int i=0; relatedTopics[i]; ++i) {
    const char *ch = relatedTopics[i];
    while(*ch) {
        putchar(*ch++);
        putchar('\n');
    }
    putchar('\n');
}
```

2 Visitor

Methode `forEach` des arrays en JS:

```
var nope = ["==", "with", "eval"];
nope.forEach((e) => {
    console.log(e)
});
```

"L'iterator" `each` de Ruby:

```
popularProjects = ["RoR", "rails", "Ruby on Rails"];
popularProjects.each do |i|
   puts i
end
});
```

3 Comment Python fait marcher tout ça

```
In [2]: cris = ['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo']

        # cris = set(['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo'])
        # cris = tuple(['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo'])
        # cris = dict.fromkeys(['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo'])

        print(f"{type(cris)} de cris :")

        for cri in cris:
            print(cri, '!')

<class 'list'> de cris :
Cowabunga !
Yippee ki-yay, mofo !
Wololo !


In [3]: cris = ['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo']

        #cris = set(['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo'])
        #cris = tuple(['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo'])
        #cris = dict.fromkeys(['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo'])

In [4]: sauce_magique = iter(cris)

In [5]: type(sauce_magique)

Out[5]: list_iterator

In [6]: next(sauce_magique)
```

```
Out[6]: 'Cowabunga'

In [7]: next(sauce_magique)

Out[7]: 'Yippee ki-yay, mofo'

In [8]: next(sauce_magique)

Out[8]: 'Wololo'

In [10]: next(sauce_magique)


        ---------------------------------------------------------------------------

        StopIteration                             Traceback (most recent call last)

        <ipython-input-10-1d04ef51b335> in <module>()
    ----> 1 next(sauce_magique)


        StopIteration:
```

4 Iterator, l'interface universelle

```
In [11]: for i in range(1, 4):
             print(i)

1
2
3


In [12]: for lettre in "abc":
             print(lettre)

a
b
c


In [13]: for ligne in open('answers.txt'):
             print(ligne)

Chez moi ça marche

T'as rebooté ?

Réessaye pour voir
```

```
In [14]: list(open('answers.txt'))

Out[14]: ['Chez moi ça marche\n', "T'as rebooté ?\n", 'Réessaye pour voir\n']

In [15]: tuple(range(1, 4))

Out[15]: (1, 2, 3)

In [16]: set('abc')

Out[16]: {'a', 'b', 'c'}

In [17]: sum([1, 2, 3])

Out[17]: 6

In [18]: sum(set([1, 2, 3]))

Out[18]: 6

In [19]: sum(range(1, 4))

Out[19]: 6

In [20]: sum(map(int, "123"))

Out[20]: 6

In [21]: sorted(set(['Cowabunga', 'Yippee ki-yay, mofo', 'Wololo']))

Out[21]: ['Cowabunga', 'Wololo', 'Yippee ki-yay, mofo']

In [22]: list(filter(bool, (True, False, None, 1, 0)))

Out[22]: [True, 1]

In [23]: "//////////////".join(open('answers.txt'))

Out[23]: "Chez moi ça marche\n//////////////T'as rebooté ?\n//////////////Réessaye pour voir\n"

In [24]: for i, ligne in enumerate('abc', 1):
             print(i, ligne)

1 a
2 b
3 c
```

4.1 L'unpacking

```
In [25]: x, y = (1, 2)
         print(x)
```

```
1
```

```
In [26]: def point():
             return 1, 2

         x, y = point()
         print(y)
```

```
2
```

```
In [27]: a, b, c = ["Aligator", "Behemot", "Claude"]
         print(f'a = {a}')
         print(f'b = {b}')
         print(f'c = {c}')
```

```
a = Aligator
b = Behemot
c = Claude
```

```
In [28]: a, b, c = open('answers.txt')
         print(f'a = {a}')
         print(f'b = {b}')
         print(f'c = {c}')
```

```
a = Chez moi ça marche

b = T'as rebooté ?

c = Réessaye pour voir
```

```
In [29]: a, *b, d = range(100)
         print(f'a = {a}')
         print(f'b = {b}')
         print(f'd = {d}')
```

```
a = 0
b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
d = 99
```

```
In [30]: print(*'abc', sep="#")  # équivaut à print("a", "b", "c", sep="#")
```

```
a#b#c
```

```
In [31]: [*range(3), *open('answers.txt'), *set(cris)]  # marche aussi pour merger des dictionna
```

```
Out[31]: [0,
          1,
          2,
          'Chez moi ça marche\n',
          "T'as rebooté ?\n",
          'Réessaye pour voir\n',
          'Cowabunga',
          'Yippee ki-yay, mofo',
          'Wololo']
```

```
In [32]: pixels = ["rouge", "vert", "bleu", "rouge", "vert", "bleu", "rouge", "vert", "bleu"]
         pixels[::3], pixels[2::3] = pixels[2::3], pixels[::3]
         pixels
```

```
Out[32]: ['bleu', 'vert', 'rouge', 'bleu', 'vert', 'rouge', 'bleu', 'vert', 'rouge']
```

```
In [79]: import collections # amis des itérables

         collections.Counter('dkjflchldkfhjdlskqfvdkfdhqkj')

         # collections.deque
         # collections.defaultdict
         # collections.OrderedDict
         # collections.namedtuple
```

```
Out[79]: Counter({'c': 1,
                  'd': 5,
                  'f': 4,
                  'h': 3,
                  'j': 3,
                  'k': 5,
                  'l': 3,
                  'q': 2,
                  's': 1,
                  'v': 1})
```

```
In [36]: # Prend n'importe quel itérable

         print(any([True, True, False, True, False]))
         print(all([True, True, False, True, False]))
         print(max([5, 10, 4, 1]))
         # min
         # reduce
```

```
True
False
10
```

Retourne des itérables:

- bytes
- csv.reader
- os.walk
- multiprocessing.Pool.map
- sqlite3.cursor
- xml.etree.ElementTree

Et même les libs externes:

- Querysets des ORMs (Django, SQLAlchemy, Peewee...);
- Body des réponses de WSGI

4.2 Un module juste pour les itérables

```
import itertools

itertools.accumulate
itertools.chain
itertools.combinations
itertools.combinations_with_replacement
itertools.compress
itertools.count
itertools.cycle
itertools.dropwhile
itertools.filterfalse
itertools.groupby
itertools.islice
itertools.permutations
itertools.product
itertools.repeat
itertools.starmap
itertools.takewhile
itertools.tee
itertools.zip_longest
```

```
In [39]: import itertools
         for x in itertools.chain('abc', range(3)):
             print(x)

a
b
c
```

```
0
1
2
```

```
In [40]: list(itertools.product('abc', range(3)))
```

```
Out[40]: [('a', 0),
          ('a', 1),
          ('a', 2),
          ('b', 0),
          ('b', 1),
          ('b', 2),
          ('c', 0),
          ('c', 1),
          ('c', 2)]
```

```
In [41]: histoire_de_la_vie = iter(itertools.cycle('abc'))
         print(next(histoire_de_la_vie))
         print(next(histoire_de_la_vie))
         print(next(histoire_de_la_vie))
         print(next(histoire_de_la_vie))
```

```
a
b
c
a
```

5 Le flux de données

```
In [42]: import subprocess

         res = subprocess.check_output(["ifconfig"], encoding="utf8")   # devinez ce qu'attend '

         print(res)
```

```
docker0   Link encap:Ethernet  HWaddr 02:42:70:a2:48:86
          inet adr:172.17.0.1  Bcast:0.0.0.0  Masque:255.255.0.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          Packets reçus:0 erreurs:0 :0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:0
          Octets reçus:0 (0.0 B) Octets transmis:0 (0.0 B)

enx644bf0013110 Link encap:Ethernet  HWaddr 64:4b:f0:01:31:10
          inet adr:192.168.1.10  Bcast:192.168.1.255  Masque:255.255.255.0
          adr inet6: 2a01:cb1d:16:5b00:6a2c:d680:372e:a92a/64 Scope:Global
```

```
          adr inet6: fe80::1ff5:8827:4cd9:fdf2/64 Scope:Lien
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Packets reçus:310258 erreurs:0 :0 overruns:0 frame:0
          TX packets:185186 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:337432415 (337.4 MB) Octets transmis:20810130 (20.8 MB)

lo        Link encap:Boucle locale
          inet adr:127.0.0.1  Masque:255.0.0.0
          adr inet6: ::1/128 Scope:Hôte
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          Packets reçus:30249 erreurs:0 :0 overruns:0 frame:0
          TX packets:30249 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:18551108 (18.5 MB) Octets transmis:18551108 (18.5 MB)

wlp2s0    Link encap:Ethernet  HWaddr 44:1c:a8:e2:1a:df
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          Packets reçus:10400 erreurs:0 :121 overruns:0 frame:0
          TX packets:5860 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 lg file transmission:1000
          Octets reçus:11993856 (11.9 MB) Octets transmis:592176 (592.1 KB)
```

```
In [43]: import re, subprocess

         res = subprocess.check_output(["ifconfig"], encoding="utf8")
         reg = re.compile(r'(\S+).*\n.*adr:(\S+)')
```

```
In [44]: interfaces = {}
         for block in res.split('\n\n'):
             if 'adr' in block:
                 interface, ip = reg.match(block).groups()
                 interfaces[interface] = ip
         print(interfaces)
```

```
{'docker0': '172.17.0.1', 'enx644bf0013110': '192.168.1.10', 'lo': '127.0.0.1'}
```

```
In [45]: interfaces = [reg.match(block).groups() for block in res.split('\n\n') if 'adr' in bloc
         dict(interfaces)
```

```
Out[45]: {'docker0': '172.17.0.1', 'enx644bf0013110': '192.168.1.10', 'lo': '127.0.0.1'}
```

5.1 Les intensions

```
In [46]: [x * x for x in range(10)]
```

```
Out[46]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

In [47]: {x: x * x for x in range(10)}

Out[47]: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

In [48]: {x * x for x in range(10)}

Out[48]: {0, 1, 4, 9, 16, 25, 36, 49, 64, 81}
```

5.2 Les expressions génératrices

```
In [49]: # NOPE !
         # res = [x * x for x in range(10000000000000)]

In [50]: res = (x * x for x in range(10000000000000))

In [51]: res

Out[51]: <generator object <genexpr> at 0x7f0808755360>

In [52]: next(res)

Out[52]: 0

In [53]: next(res)

Out[53]: 1

In [54]: print(next(res))
         print(next(res))
         print(next(res))

4
9
16


In [55]: import sqlite3

         def get_results(n=10, profile="/home/pycon/.mozilla/firefox/l92ue2kx.default/places.sql
             return sqlite3.connect(profile).execute("""
                 SELECT sites.rev_host as host, count(*) as visits FROM moz_historyvisits as vis
                 WHERE  visits.place_id == sites.id GROUP BY host ORDER BY visits DESC
             """)

In [56]: sites = ((dom[-2::-1], vis) for dom, vis in get_results())

In [57]: MOTEURS = set(['duckduckgo', 'google', 'bing', 'qwant'])
         sites = ((dom, vis) for dom, vis in sites if not any(m in dom for m in MOTEURS))
```

```
In [58]: import itertools
         sites = itertools.islice(sites, 0, 5)

In [59]: sites

Out[59]: <itertools.islice at 0x7f080873adb8>

In [60]: next(sites)

Out[60]: ('localhost', 359)

In [61]: next(sites)

Out[61]: ('nbviewer.jupyter.org', 41)

In [62]: for s in sites:
             print(s)

('github.com', 9)
('lastpass.com', 9)
('www.lastpass.com', 7)
```

5.3 Yield

```
In [63]: def fonction_normale():
             print('Avant le premier return')
             return 1
             print('Apres le premier return')
             return 2
             print("Apres le second return")
             return 3
             print('Tout à la fin')

         res = fonction_normale()
         print(res)

Avant le premier return
1
```

```
In [64]: def fabriquer_un_generateur():
             print('Avant le premier yield')
             yield 1
             print('Apres le premier yield')
             yield 2
             print("Apres le second yield")
             yield 3
             print('Tout à la fin')

         res = fabriquer_un_generateur()
         print(res)
```

```
<generator object fabriquer_un_generateur at 0x7f0808755570>
```

```
In [65]: next(res)
```

```
Avant le premier yield
```

```
Out[65]: 1
```

```
In [66]: next(res)
```

```
Apres le premier yield
```

```
Out[66]: 2
```

```
In [67]: for x in res:
             print(x)
```

```
Apres le second yield
3
Tout à la fin
```

```
In [68]: import secrets

         def secret_key_generator(n):
             for i in range(n):
                 yield secrets.token_hex()

         gen = secret_key_generator(10)
```

```
In [69]: next(gen)
```

```
Out[69]: '9c814af4bceb652403b32d908973ae10c9433e6031a51cb0221f7196c5420aea'
```

```
In [70]: next(gen)
```

```
Out[70]: '6acd1731134c0b8db62f42bab49652c4accf91731dd9e1c00d1fdf2f5ee28f5d'
```

```
In [71]: import itertools

         class MachineAtuer:
             def __iter__(self):
                 while True:
                     yield "Miaou !"

         for x in itertools.islice(MachineAtuer(), 10):
             print(x)
```

```
Miaou !
Miaou !
Miaou !
Miaou !
Miaou !
Miaou !
Miaou !
Miaou !
Miaou !
Miaou !
```

5.4 Tous les tuyaux mis bouts à bouts

```
In [72]: import string
         from pathlib import Path


         def lister_mots_cles(dossier, ext):
             for chemin in Path(dossier).glob(f'./**/*.{ext}'): # iterable
                 try:
                     with open(chemin) as f:
                         for ligne in f: # iterable
                             for mot in ligne.split(): # iterable
                                 # expression generatrice sur un itérable passée à join... qui a
                                 mot = "".join(l for l in mot if l not in string.punctuation).st
                                 if mot:
                                     # génération de données
                                     yield mot
                 except Exception:
                     pass

In [73]: from collections import Counter

         # Counter accepte un itérable en a paramètre
         for mot, score in Counter(lister_mots_cles('/etc', 'conf')).most_common(5): # itérable
             print(f'- {mot}: {score}')
```

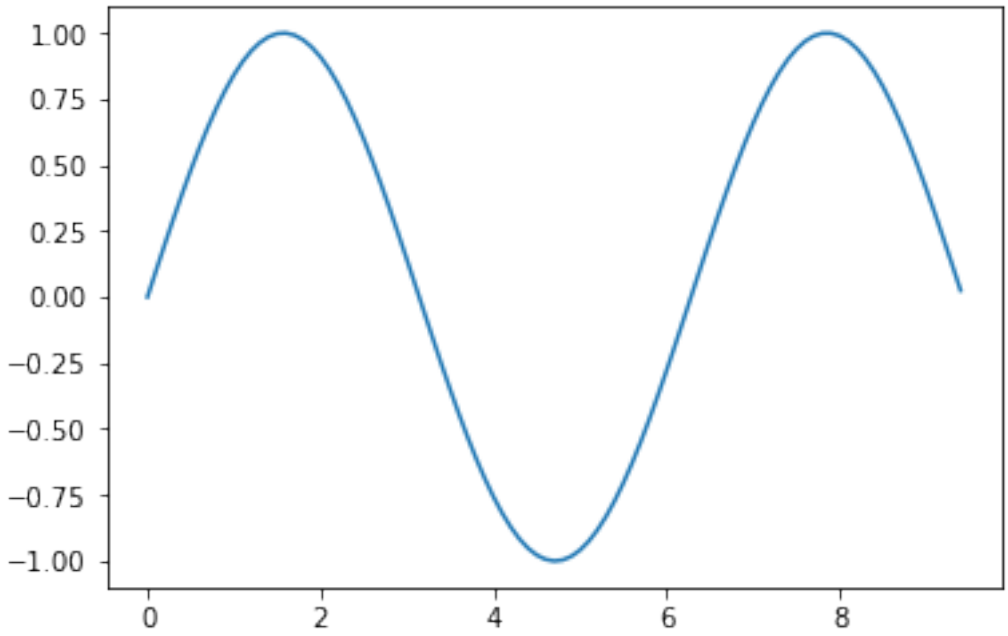
```
- the: 2937
- alias: 1504
- to: 1477
- for: 1100
- if: 984
```

```
In [74]: import numpy as np # ou scipy, pandas, etc

         import matplotlib.pyplot as plt
```

```
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

plt.plot(x, y)
plt.show()
```




```
In [75]: import math

         x = np.arange(0, 3 * np.pi, 0.1)

         %timeit [math.sin(i) for i in x]
         %timeit np.sin(x)
```

```
23.4 µs ś 491 ns per loop (mean ś std. dev. of 7 runs, 10000 loops each)
2.5 µs ś 36.7 ns per loop (mean ś std. dev. of 7 runs, 100000 loops each)
```

6 Il y a un "one more thing" pour ça

```
In [76]: def crieur():
             while True:
                 res = yield
                 print(res.upper(), '!!')

         gen = crieur()
         next(gen)
```

```
In [77]: interdits = [
             'les interdictions',
             'les listes',
             'les interdictions',
             'les repetitions',
             'les mets',
             'les blagues faciles'
         ]

         for x in interdits:
             gen.send(x)
```

```
LES INTERDICTIONS !
LES LISTES !
LES INTERDICTIONS !
LES REPETITIONS !
LES METS !
LES BLAGUES FACILES !
```

Télécharger les slides
https://huit.re/philo-iteration-fr