La philosophie de l'it?ration

September 22, 2017

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

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

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 [ ]: 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, '!')
```

```
In [ ]: 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 [ ]: sauce_magique = iter(cris)
```

```
In [ ]: type(sauce_magique)
```

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

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

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

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

4 Iterator, l'interface universelle

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

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

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

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

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

In [ ]: set('abc')

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

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

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

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

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

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

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

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

4.1 L'unpacking

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

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

        x, y = point()
        print(y)

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

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

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

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

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

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

In [ ]: import collections # amis des itérables

        Counter('dkjflch ldkfhjdlskqfvdkfdhqkj')

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

In [ ]: # 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
```

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

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

```
In [ ]: 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))
```

5 Le flux de données

```
In [ ]: import subprocess

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

        print(res)
```

```
In [ ]: import re, subprocess

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

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

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

5.1 Les intensions

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

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

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

5.2 Les expressions génératrices

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

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

In [ ]: res

In [ ]: next(res)

In [ ]: next(res)

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

In [ ]: import sqlite3

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

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

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

In [ ]: sites

In [ ]: next(sites)

In [ ]: next(sites)

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

5.3 Yield

```
In [ ]: 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)
```

```
In [ ]: 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)
```

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

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

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

```
In [ ]: import secrets

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

        gen = secret_key_generator(10)
```

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

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

```
In [ ]: import itertools

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

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

5.4 Tous les tuyaux mis bouts à bouts

```
In [ ]: 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 at
                                mot = "".join(l for l in mot if l not in string.punctuation).str
                                if mot:
                                    # génération de données
                                    yield mot
                except Exception:
                    pass
```

```
In [ ]: 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}')
```

```
In [ ]: 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 [ ]: import math

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

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

6 Il y a un "one more thing" pour ça

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

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

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

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

Télécharger les slides
https://formationspython.com/philo-iteration-fr.zip