Générateurs en Python, un exemple réel

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 2 juin 2009

https://www.bortzmeyer.org/generateurs-python.html

Les fanas du langage de programmation Python citent souvent parmi les merveilles de ce langage la possibilité d'écrire des **générateurs**, c'est-à-dire des fonctions qui gardent automatiquement un état d'un appel sur l'autre et peuvent servir à énumérer à loisir une structure de données. Beaucoup d'exemples de générateurs qu'on trouve dans les tutoriels sont plutôt artificiels et, en les lisant, on se dit « C'est une idée amusante mais à quoi peut-elle servir en vrai ? ». Voici donc un exemple de générateur tiré d'un programme réel.

Le problème était d'analyser le journal d'un serveur whois. Pour chaque requête d'un client whois, on va mettre une information dans une base de données. Ce motif de conception est courant en administration système et, sans les générateurs, se programme en général ainsi (en pseudo-code) :

```
for each line in log
    entry = parse(line)
    do something for the entry
```

avec mélange de l'analyse syntaxique (`parse(line)`) et de l'action effectuée (`do something for the entry`). Or, ces deux calculs n'ont rien à voir et n'ont aucune raison d'être imbriqués dans la même boucle, d'autant plus qu'on pourrait parfaitement vouloir les utiliser séparement. Les générateurs permettent de séparer les deux calculs dans des endroits bien distincts du programme :

```
# Le générateur
def whois_entry
    for each line in log
       entry = parse(line)
       yield entry

for each entry in whois_entry()
    do something for the entry
```

En Python, le mot-clé `yield` renvoie une valeur (comme `return`) mais la fonction ne se termine pas. Le prochain appel continuera à l'instruction suivant `yield` et la boucle sur le journal reprendra au point où elle s'était arrêtée.

Les générateurs ont été, à ma connaissance, introduits par le langage CLU et très bien présentés dans le livre de Liskov, « *"Abstraction and Specification in Program Development"* ». Ce livre date des années 1980, ce qui donne une idée du temps qu'il faut en informatique pour qu'une idée devienne banale. Pour les générateurs Python, une très bonne introduction existe dans Wikipédia.

Enfin, voici le code presque complet qui analyse le journal du serveur whois et met le résultat dans une base de données :

```
#!/usr/bin/python

import re
import sys
import psycopg2

db = "dbname=essais"

def parse_whois(filename):
    ifile = open(filename)
    for line in ifile:
        match = whois_request.search(line)
        if not match:
            print >>sys.stderr, "Warning: invalid line \"%s\"" % line
            continue
        time = match.group(2) + ":" + match.group(3) + ":" + match.group(4)
        source = match.group(6)
        request = match.group(8)
        if request.find(' ') == -1:
            domain = request
        else:
            args = request.split(' ')
            domain = args[len(args)-1]
        yield (time, source, unicode(domain.lower(), "latin-1"))

whois_request = re.compile("^\[(\d+)-(\d+)H(\d+):(\d+)\]\s+Request from \[([\da-f\.:/]+)\] +\[([\da-f\.:]+)/

if __name__ == '__main__':
    filename = sys.argv[1]
    ...
    conn = psycopg2.connect(db)
    cursor = conn.cursor()
    cursor.execute("BEGIN;")
    ...
    for (time, source, request) in parse_whois(filename):
            date = day + ":" + time
            cursor.execute("""INSERT INTO whois_requests (file, seen, source, domain)
                              VALUES (%(file_id)s, %(date)s,
              %(source)s, %(request)s);", locals())
    cursor.execute("COMMIT;")
```