RECETTE 75

MESURER LE TEMPS D'EXÉCUTION d'une commande

L'OBJECTIF

Nous disposons de deux implémentations d'une fonction factorielle et nous souhaitons déterminer laquelle est la plus efficace.

```
01: def fact_it(n):
02:     result = 1
03:
04:     for i in range(n):
05:         result += result * i
06:
07:     return result
08:
09: def fact_rec(n):
10:     if n == 0 or n == 1:
11:         return 1
12:     else:
13:         return n * fact_rec(n - 1)
14:
15: if __name__ == '__main__':
16:     print('{}! = {}'.format(6, fact_it(6)))
17:     print('{}! = {}'.format(6, fact_rec(6)))
```

LA SOLUTION

```
01: import timeit
...
17: if __name__ == '__main__':
...
20:     tps_fact_it = timeit.repeat('fact_it(32)', 'from __main__
import fact_it', repeat=3, number=10000)
21:     tps_fact_rec = timeit.repeat('fact_rec(32)', 'from __main__
import fact_rec', repeat=3, number=10000)
22:     print('Tps pour fact_it : {}'.format(min(tps_fact_it)))
23:     print('Tps pour fact_rec : {}'.format(min(tps_fact_rec)))
```

COMMENTAIRES

Le module **timeit** fournit des fonctions pour évaluer le temps d'exécution d'un code. `repeat()` permet d'exécuter `repeat` mesures de la commande passée en premier paramètre et exécutée `number` fois. Le résultat de l'exécution de notre code est le suivant :

```
6! = 720
6! = 720
Tps pour fact_it : 0.040213363001385005
Tps pour fact_rec : 0.05365261499900953
```

NOTE

Le deuxième paramètre de `repeat()` permet d'importer la fonction que nous voulons tester et qui a été définie dans l'espace de nom courant (nommé `__main__`). Ce paramètre est inutile si l'on désire tester des fonctions standards de Python :

```
timeit.repeat('liste = [2, 3, 4]', repeat=3, number=10000)
```

Vous pouvez également lancer directement la commande suivante :

```
$ python -m timeit 'liste = [2, 3, 4]'
```

Vous pouvez également lancer directement la commande suivante :

```
$ python3 -m timeit 'liste = [2, 3, 4]'
```

Les options disponibles sont :

- `-u` permettant de spécifier l'unité de temps (µsec (`usec`), `msec` ou `sec`). Par exemple :

```
$ python3 -m timeit -u usec 'liste = [2, 3, 4]'
10000000 loops, best of 3: 0.0582 usec per loop
$ python3 -m timeit -u msec 'liste = [2, 3, 4]'
10000000 loops, best of 3: 5.81e-05 msec per loop
```

- `-s` pour indiquer un traitement à exécuter une seule fois au départ de la série de mesures. Par exemple :

```
$ python3 -m timeit -s 'f=lambda x : x**2'  'f(10)'
1000000 loops, best of 3: 0.33 usec per loop
```

- `-n` pour déterminer le nombre de fois où le traitement devra être exécuté. Par exemple :

```
$ python3 -m timeit -n 100 'liste = [2, 3, 4]'
100 loops, best of 3: 0.119 usec per loop
```

- `-r` qui permet d'indiquer le nombre de fois où la série de mesures doit être lancée. Par exemple :

```
$ python3 -m timeit -r 4 'liste = [2, 3, 4]'
10000000 loops, best of 4: 0.0588 usec per loop
```

Si vous utilisez l'interpréteur IPython, la commande magique `%timeit` permet d'évaluer le temps d'exécution d'une commande directement depuis le shell IPython :

```
$ ipython
In [1]: %timeit liste=[1, 2, 3]
The slowest run took 19.07 times longer than the fastest. This
could mean that an intermediate result is being cached.
10000000 loops, best of 3: 68.9 ns per loop
```