code_liste_tuple

April 14, 2021

1 1A.1 - Liste, tuple, ensemble, dictionnaire, liste chaînée, coût des opérations

Exemples de containers, list, tuple, set, dict.

[1]:
```
from jyquickhelper import add_notebook_menu
add_notebook_menu()
```

[1]:
```
<IPython.core.display.HTML object>
```

Python propose différents containers pour stocker des éléments. Voici les plus courants :

- list : tableau d'éléments indexés de 0 à n exclu auquel on peut ajouter ou retirer des éléments
- dict : tableau d'éléments indexés par des types immuables auquel on peut ajouter ou retirer des éléments
- tuple : tableau d'éléments indexés de 0 à n exclu qu'on ne peut pas modifier
- set : tableau d'éléments uniques non indexés
- frozenset : `set` immuables (non modifiable)
- deque : presque équivalent à une listes, la différent vient de l'implémentation, les mêmes opérations n'auront pas les mêmes coûts (deque = liste chaînée)

D'autres containers sont disponibles via le module collections. Tous proposent de stocker un nombre variables d'éléments. Deux aspects diffèèrent :

- la façon de désigner un élément de l'ensemble
- le coût de certaines opérations, il faut choisir qui minisera le coût des opérations pour votre programme

1.0.1 Insertion avec `list` et `deque`

On veut comparer les coûts d'insertion en début et fin de liste pour un grand nombre d'éléments.

[2]:
```
import time, collections
N = 1000000

for p in range(0,3):
    print("passage ", p)
    print("  insertion en fin")

    li = list()
    a = time.perf_counter()
    for i in range(0,N) :
        li.append(i)
    b = time.perf_counter()
```

```
    print("    list", N, "éléments, temps par éléments :", (b-a)/N)

    li = collections.deque()
    a = time.perf_counter()
    for i in range(0,N) :
        li.append(i)
    b = time.perf_counter()
    print("    deque", N, "éléments, temps par éléments :", (b-a)/N)

    print("  insertion au début")
    li = collections.deque()
    a = time.perf_counter()
    for i in range(0,N) :
        li.appendleft(i)
    b = time.perf_counter()
    print("    deque", N, "éléments, temps par éléments :", (b-a)/N)

    N2 = N // 100
    li = list()
    a = time.perf_counter()
    for i in range(0,N2) :
        li.insert(0,i)
    b = time.perf_counter()
    print("    list", N, "éléments, temps par éléments :", (b-a)/N)
```

```
passage  0
  insertion en fin
    list 1000000 éléments, temps par éléments : 1.6768032881764787e-07
    deque 1000000 éléments, temps par éléments : 1.3930898200138983e-07
  insertion au début
    deque 1000000 éléments, temps par éléments : 1.2026622409235594e-07
    list 1000000 éléments, temps par éléments : 2.2802558892816237e-08
passage  1
  insertion en fin
    list 1000000 éléments, temps par éléments : 1.5078710092361442e-07
    deque 1000000 éléments, temps par éléments : 1.210950632710861e-07
  insertion au début
    deque 1000000 éléments, temps par éléments : 1.2431481508550512e-07
    list 1000000 éléments, temps par éléments : 2.6522458657794125e-08
passage  2
  insertion en fin
    list 1000000 éléments, temps par éléments : 1.4364004201874892e-07
    deque 1000000 éléments, temps par éléments : 1.3574118094175568e-07
  insertion au début
    deque 1000000 éléments, temps par éléments : 1.3737357535858763e-07
    list 1000000 éléments, temps par éléments : 2.607208846534781e-08
```

On voit que l'insertion au début du tableau est beaucoup plus coûteuse pour une liste que pour un `deque`.

1.0.2 Un élément dans un ensemble

Faut-il écrire `i in [0,1]` ou `i in (0,1)` ou … Essayons.

[3]:
```
import time, collections
N = 100000
lens = list(range(0,1000))
tens = tuple(lens)
sens = set(lens)
fens = frozenset(lens)

for p in range(0,3):
    print("passage",p)
    a = time.perf_counter()
    s = 0
    for i in range(0,N) :
        if i in lens : s += 1
    b = time.perf_counter()
    print("  list", N, "fois, temps par éléments :", (b-a)/N)

    a = time.perf_counter()
    s = 0
    for i in range(0,N) :
        if i in tens : s += 1
    b = time.perf_counter()
    print("  tuple", N, "fois, temps par éléments :", (b-a)/N)

    a = time.perf_counter()
    s = 0
    for i in range(0,N) :
        if i in sens : s += 1
    b = time.perf_counter()
    print("  set", N, "fois, temps par éléments :", (b-a)/N)

    a = time.perf_counter()
    s = 0
    for i in range(0,N) :
        if i in fens : s += 1
    b = time.perf_counter()
    print("  frozenset", N, "fois, temps par éléments :", (b-a)/N)
```

```
passage 0
  list 100000 fois, temps par éléments : 1.1804175089708608e-05
  tuple 100000 fois, temps par éléments : 1.2459357053093508e-05
  set 100000 fois, temps par éléments : 8.682663236478483e-08
  frozenset 100000 fois, temps par éléments : 9.215601297539955e-08
passage 1
  list 100000 fois, temps par éléments : 1.1412313516123058e-05
  tuple 100000 fois, temps par éléments : 1.1940795282648774e-05
  set 100000 fois, temps par éléments : 8.101922725166411e-08
  frozenset 100000 fois, temps par éléments : 8.893231054526218e-08
passage 2
  list 100000 fois, temps par éléments : 1.1539950008908635e-05
  tuple 100000 fois, temps par éléments : 1.1629893677079037e-05
  set 100000 fois, temps par éléments : 8.061231383216239e-08
  frozenset 100000 fois, temps par éléments : 9.595650530114242e-08
```

Il apparaît que les ensemble `set` ou `frozenset` sont beaucoup plus rapides. Plus l'ensemble est grand, plus

cette différence est importante.

[4]: