Haute École Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

2023 – 2024
DEV1 – DEV1L

DEV1 – DEV1L – Laboratoires Python

TD 11 – Tuples et listes

Ce TD a pour objectif de présenter les tuples (*tuples*) et les listes (*lists*), s'exercer à les utiliser et les différencier.

Table des matières

1 Les tuples - `()`

Un tuple est un ensemble de valeurs portant le même nom et accessibles par un indice. Un tuple se crée en utilisant les *parenthèses* (`()`).

Exemples :

```
tuple_1 = (1, 2, 3, 4, 5)
tuple_2 = tuple(range(10_000_000))
```

- `tuple_1` contient les valeurs 1, 2, 3, 4 et 5 qui sont accessibles par un indice à partir de 0. Par exemple `tuple_1[0]` vaut 1.
- `tuple_2` contient les valeurs de 0 à 9 999 999 et `tuple_2[1_000_000]` contient la valeur 1 000 000.

Astuce

Il est autorisé — c'est plus lisible — de séparer les chiffres d'une valeur entière par le caractère « barre de soulignement » (*underscore*).
Ainsi au lieu d'écrire `10000000`, il est possible d'écrire `10_000_000`.

Version du document
12 novembre 2023

Exercice 1 Maximum

Soit le tuple suivant : `(-4, 12, -71, 33, 20, 32, 96, -22, -7, 70, 82, 62, 11, 72, -36, -16, 84)`. Écrivez une fonction `ya_max` qui recherche la valeur maximum de ce tuple.

Affichez le résultat de votre fonction de recherche de maximum et de la fonction *built in* (`max`).

Exercice 2 Temps d'accès moyen

Écrivez un programme qui affiche le temps d'accès [1] moyen à une valeur d'un tuple lors d'un parcours.

Accéder à une valeur d'un tuple se fait grâce aux crochets. J'accède à la première valeur du tuple `a_tuple` par `a_tuples[0]`.

Pour calculer le temps d'accès à une valeur, je note le temps courant [2] avant l'accès et après l'accès. La différence des deux me donne le temps d'accès. Par exemple :

```
start_time = time.time()
for j in range(len(a_tuple)):
    value = a_tuple[j]
end_time = time.time()
```

...le temps d'accès est de `end_time - start_time`.

Créez un tuple contenant les valeurs de 0 à 4 999 999 [3] et calculez la somme des éléments du tuple ainsi qu'une estimation du temps utilisé pour calculer cette somme.

Faites 50 fois ce calcul et affichez la moyenne des temps estimés.

Une exécution du programme peut ressembler à :

```
12499997500000 0.7167608738s
12499997500000 0.7248282433s
[cut]
12499997500000 0.7123718262s
Average:  0.7316267633s
```

1.1 Immuabilité

Un tuple est **immuable** (*immutable*) c'est-à-dire que ses valeurs — et sa taille — ne peuvent changer.

```
tuple_1[0] = -1 # ERROR
```

Il n'existe pas de fonction permettant l'ajout d'une valeur à un tuple. Par contre, l'opérateur `+` concatène deux tuples en retournant un *nouveau* tuple contenant les valeurs des deux opérandes. L'opérateur `*` (appliqué à un tuple et un entier n) retourne, quant-à lui un nouveau tuple contenant n fois les valeurs du tuple multiplié.

1.2 Tranches (*slicing*)

Les tuples [4] peuvent être découpés en **tranches** (*slicing*) : `a[i:j]` sélectionne tous les éléments d'indice k tel que $i <= k < j$ et crée un nouveau tuple (immuable).

1. Le temps d'accès est la durée que met la machine pour accéder à cette zone mémoire et retourner la valeur qui s'y trouve. Si le tuple est grand, ce temps peut devenir non négligeable.
2. Le module *time* fournit une fonction `time` qui retourne le moment courant.
3. La fonction `range` sera utile.
4. Et toutes les séquences d'ailleurs : string, tuples, listes...

Quand on l'utilise dans une expression, la tranche est du même type que la séquence. Ceci veut dire que l'ensemble des indices de la tranche est renuméroté de manière à partir de 0.

```
tuple_1 = ('a', 'b', 'c', 'd', 'e')
tuple_3 = tuple_1[2:4]
print(tuple_3)  # ('c', 'd')
```

Exercice 3 **En une seule instruction**

Soit le tuple suivant : `a_tuple = (15, 18, 12, 14, 20, 10, 16, 19, 9, 13)`. En une seule instruction, affichez ce tuple trié [5] sans la plus petite valeur ni la plus grande.

Le (tout petit) programme affichera donc `(10, 12, 13, 14, 15, 16, 18, 19)` [6].

2 Les listes - []

Une liste est un ensemble de valeurs portant le même nom et accessibles par un indice. Une liste se crée en utilisant les *crochets* (`[]`) [7].

Exemples :

```
list_1 = [1, 2, 3, 4, 5]
list_2 = list(range(10_000_000))
```

Exercice 4 **Temps d'accès moyen**

Reprendre l'exercice 2 page précédente avec une liste cette fois et comparer les valeurs obtenues.

La différence entre un tuples et une liste, c'est qu'une liste peut être modifiée. Elle est mutable.

2.1 Mutabilité

Une liste est **mutable** (*mutable*) c'est-à-dire que ses valeurs — et sa taille — peuvent changer.

La fonction `append` ajoute un élément à la liste.

```
list_1[0] = -1
list_1.append(6)  # [-1, 2, 3, 4, 5, 6]
```

La fonction `extend` ajoute les valeurs d'une deuxième liste à la première.

```
list_3 = [7, 8, 9]
list_1.extend(list_3)
print(list_1)  # [-1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Attention

Là où `extend` *ajoute* des éléments à une liste, l'opérateur `+` crée une nouvelle liste.

Exercice 5 **extend versus +**

Écrivez un exemple pertinent montrant la différence entre l'opérateur `+` pour les listes et la fonction `extend`.

5. Nous avons déjà vu une instruction pour trier.
6. Pour obtenir un tuple je dois copier la liste obtenue pour créer un nouveau tuple... ce qui n'est pas très utile ni efficace. Si votre programme affiche une liste, c'est très bien aussi.
7. Pour rappel, vous avez déjà rencontré les listes dans les tds précédents

3 Exercice - Une seule ligne

Écrivez un petit jeu qui se joue comme suit :

- ▷ le jeu est une suite de 20 cases pouvant chacune contenir une valeur comprise entre 1 et 4 ;
- ▷ au début du jeu seules deux cases contiennent une valeur qui aura été choisie aléatoirement ;
- ▷ à chaque tour de jeu, l'ordinateur ajoute deux valeurs aléatoires [8] à deux endroits libres choisis aussi aléatoirement ;
- ▷ à chaque tour de jeu, la ou le joueur doit déplacer une valeur vers la gauche ou vers la droite d'autant de cases qu'il ou elle le désire tant que toutes les cases entre la position de départ et celle d'arrivée sont libres ;
- ▷ dès que 3 cases consécutives contiennent la même valeur, elles sont vidées ;

Le but du jeu est de jouer le plus de coups possibles avant que les 20 cases ne soient remplies. Lorsqu'il n'est plus possible de jouer, le programme affiche le nombre de tours joués.

Voici un exemple de début de partie :

```
INLINE
                        2                                       2
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
Enter origin and destination : 6 15
                        3               3                   2   2
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
Enter origin and destination : 10 7
  4                 1   3   3                               2   2
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
Enter origin and destination : 5 3
  4           1       1   3   3                               2   2           2
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
Enter origin and destination : 19 17
  4   2       1       1   3   3                           1
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
```

Voici un peu d'aide pour résoudre l'exercice :

1. créez une liste de 20 éléments initialisés à 0 ;
2. écrivez une fonction `add_2_values()` qui ajoute deux valeurs à une place aléatoire et de valeur aléatoire (pas spécialement différentes) comprises entre 1 et 4 ;
3. écrivez une fonction `display()` qui affiche cette liste ;
4. écrivez une fonction `move(origin, destination)` qui déplace la valeur de la position d'origine vers la position de destination. Attention, les cases entre les deux et la destination doivent être libres ;
5. écrivez une fonction `read()` qui lit les deux valeurs. Cette fonction peut retourner un *tuple* et doit être un peu robuste ;
6. écrivez une fonction `remove_3_inline()` qui cherche s'il y a 3 valeurs identiques et consécutives et qui les supprime s'il échet ;
7. écrire la fonction principale qui devrait se résumer à une boucle qui ; affiche, lit les valeurs, déplace, ajoute 2 valeurs, supprime le triplet éventuel.

8. Comprises entre 1 et 4 donc