TP N°5: Les tuples

Définition

Un tuple ou *n-uplet* en Python est une *séquence ordonnée* et *non modifiable* d'éléments éventuellement *hétérogènes* séparés par des virgules et entourés de parenthèses (Les parenthèses ne sont pas obligatoires).

- Le type *tuple* est un type de données similaire au type `list`. La seule différence avec les listes est que les tuples sont *non-modifiables* (on ne peut pas modifier un élément). Utiliser un tuple à la place d'une liste revient à avoir une assertion implicite que les données sont constantes.
- Le *tuple vide* (appelé aussi *0-uplet*) se note `( )`.
- Un tuple contenant 1 élément `x` se note `(x,)` et non `(x)` qui désigne juste `x`.
- Un tuple à deux éléments (couple) x et y se note `(x,y)` (ou `(x, y,)`).
- On peut convertir une liste en tuple avec la fonction `tuple()` ainsi on a :
  `tuple ( [ 4 2 , 17 , 7 3 ] ) == (42 , 17 , 73)`

Étant donné un tuple t, on peut, avec la même syntaxe que pour les listes :

- accéder à son k – $i^{\text{ème}}$ élément avec la syntaxe `t[k]` (comme pour les listes, la numérotation commence à partir de 0 comme pour les listes,
- désigner les éléments en comptant à partir de la fin en prenant des valeurs de k négatives),
- calculer son nombre d'éléments avec `len(t)`,
- et effectuer du `tranchage` (`slicing`).

Création

```
t1 = tuple() # créer un tuple vide
t2 = (1, "ok", "non") # créer un tuple non vide
t3 = tuple([1,2,3]) # créer un tuple à partir d'un itérable (liste)
t4 = tuple(range(0,1000,200)) # créer un tuple à partir d'un itérable (intervalle)
t5 = tuple(x*x for x in range(20) if x % 3 == 0) # créer un tuple en compréhension
t6 = tuple('aeiouy') # ('o', 'i', 'e', 'a', 'y', 'u')
t7 = 1, 2, 3 # Créer un tuple sans utiliser les parenthèses
t8 = 1, # n'oubliez pas d'y ajouter une virgule, sinon ce n'est pas un tuple.
```

Remarques :

- Les parenthèses ne sont pas obligatoires mais facilite la lisibilité du code (rappelons que la force de python est sa simplicité de lecture).
- Un tuple a une longueur fixée dès sa création (on ne peut pas lui ajouter un élément sans recréer le tuple)

Fonctions et méthodes associées aux tuples

```
t = (2, 0, 5, [7, 6, 17], 4) # tuple initiale
len(t) # renvoie le nombre d'éléments d'un tuple.
del(t) # Supprimer définitivement le tuple
t1,t2 = (1, 2, 3) , (4, 5, 6)
t1 + t2 # renvoie un tuple (nouvelle séquence) résulat  de concaténation de t1 et t2
t1 * 3 # concaténer plusieurs copies du tuple t1 avec l'opérateur *.
(1,) * 3 # utile pour l'initialisation
t1 == t2 # Comparer le contenu de deux tuples
2 in t1 # tester avec in et not in si un élément appartient ou non à un tuple.
max(t1), min(t1), sum(t1) # Retourner le maximum, le minimum et la somme des éléments d'un tuple
t = (2, 1, 4, 3)
L = sorted(t) # renvoie une liste triée dont les éléments sont ceux du tuple t.
t[1] = 3 # TypeError: 'tuple' object does not support item assignment
a,b = (1,2) # affectation multiple
```

Exercice 1

1. On donne le tuple suivant : `tup=(1,3,5,'Hilbert',(12,'ipt','génial'),4.6)`
   Écrire les lignes de code qui affiche :
   ```
   Hilbert
   le dernier élément du tuple
   (3,5)
   'ipt','génial'
   ```
2. Tapez les commandes Python permattant de lire un nombre réel, séparer les parties avant et après le point décimal et les afficher sous forme de tuple. Par exemple : pour 65.21 on doit afficher `(65,21)`

Slicing

Le *slicing* (découpage en tranches en français) permet de récupérer un morceau d'un tuple :
`t[i:j]` représente le sous-tuple de `t` formé des éléments dont les indices sont compris entre `i` et `j-1`.
On peut spécifier un pas égal à `k` en écrivant `t[i:j:k]`.

```
t = (2, 8, 1, 3, 5)
t[2:4] # éléments d'indices 2 et 3
t[2:] # éléments d'indices 2 et plus (jusqu'à la fin de la liste)
t[:2] # éléments d'indices strictement inférieurs à 2
t[:] # tous les éléments de t
t[1:5:2] # éléments d'indices 1 et 3
t[::-1] # tout le tuple mais à l'envers
```

Itération sur les tuples

```
for x in  t: # On peut itérer directement sur un tuple  : x représente un élément de t
    print(x)

for i in range(len(t)): # i représente un indice variant de 0 à len(t) - 1
    print(t[i])
```

Exercice 2

1. Construire, à partie d'une nombre entier `n`, le tuple `t1` composée de ses chiffres.
   Exemple : `n = 1234` $\rightarrow$ `t1 = (1,2,3,4)`
2. Écrire un programme python qui permet de saisir un tuple `t2` de 5 entiers strictement positifs. Afficher le tuple trié en ordre descendant.
3. Écrire un programme python qui permet de saisir un tuple `t3` de n nombres et affiche le produit de ses éléments.
4. Écrire un programme python qui à partir d'une tuple d'entiers `t4`, affiche un couple (2-uplet) de tuples, le premier sous-tuple contenant les éléments pairs de `t` et le second ses éléments impairs.
   Exemple : `t4 = (1, 2, 4, 7, 3, 2, 1)` $\rightarrow$ `( (2, 4, 2), (1, 7, 3, 1) )`
5. Écrire un script python qui, étant donné un tuple `t`, renvoie le nombre maximum de zéros consécutifs dans le tuple.
   Par exemple : pour `t=(1,0,2,0,0,4)` le programme renvoie 2.
6. Écrire un script python qui teste si deux tuples ont au moins un élément commun.
7. Écrire un script python qui, étant donné un tuple `t`, crée un tuple de tous les sous-tuples possibles.
   Par exemple :
   Les sous-tuples de `(1, 2, 3)` sont : `((), (1), (2), (3), (1, 2), (2, 3), (1, 2, 3))`.