

## TP4 : listes, tuples et dictionnaires en Python

### Rappel de cours sur les listes en python

Sous Python, une liste est une collection d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets. Ce sont des objets de type «**list**» intégrant des fonctions pour manipuler les éléments qu'elles contiennent, tout comme les chaînes de caractères avec «**str**». Les éléments d'une même liste peuvent être de types différents (entiers, réels, chaînes de caractères, objets, ...).

Exemple avec l'interpréteur Python :

```
>>> jour = ['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>> print(jour)
['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
```

On peut accéder à un élément de la liste par son index (ou indice) de position dans la liste. La première position est la position «**zéro**» et la dernière correspond à «**la longueur de la liste moins un**». Il suffit d'appeler directement la liste par son nom suivi par l'indice placé entre crochets.

```
>>> print(jour[2])
mercredi
>>> print(jour[4])
20.357
```

Python permet également d'utiliser des index de position dans la liste depuis son dernier élément jusqu'au premier. Ces index sont des entiers négatifs de valeur «**-1**» pour le dernier élément jusque «**-longueur\_liste**» pour le premier

```
>>> print(jour[-1])
vendredi
>>> print(jour[-4])
1800
```

Il est possible d'utiliser les fonctions intégrées sur les listes comme «**len()**» pour afficher sa longueur, «**del()**» pour supprimer un élément.

Les fonctions de l'objet «**list**» les plus communes sont:

- la fonction «**append()**» pour ajouter un élément fournit en argument,
- la fonction «**sort()**» pour la trier,
- la fonction «**reverse()**» pour inverser ses éléments,
- la fonction «**index()**» pour retrouver un élément dont la valeur est transmise en argument
- la fonction «**remove()**» qui permet de supprimer le premier élément de la liste ayant la valeur transmise en argument.

Elles sont appelées à partir de la liste elle-même :

```
>>> print(len(jour))
7
>>> del(jour[3])
>>> print(jour)
['lundi', 'mardi', 'mercredi', 20.357, 'jeudi', 'vendredi']

>>> jour.remove(20.357)
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
>>> jour.append('samedi')
>>> print(jour)
['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi']
>>> print(jour.index('samedi'))
5
>>> jour.reverse()
['samedi', 'vendredi', 'jeudi', 'mercredi', 'mardi', 'lundi']
>>> jour.sort()
>>> print(jour)
['jeudi', 'lundi', 'mardi', 'mercredi', 'samedi', 'vendredi']
```

Exemple d'appel des méthodes de l'objet liste appliquées sur la liste 'jour'

### **Tableau multi-dimensionnel – listes de listes**

Pour construire un tableau à plusieurs dimensions, il existe plusieurs méthodes sous Python. On peut construire un tableau – une matrice, si le tableau est à 2 dimensions - comme une liste de listes :

Exemple de matrice:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
>>> a=[[1, 2] ,[3, 4] ,[5, 6]]
>>> a
[[1, 2], [3, 4], [5, 6]]
>>> len(a)
3
>>> a[1]
[3, 4]
>>> a[1][0]
3
```

Un tableau de dimension 2 (liste de listes) contiendra des éléments qui devront être appelé par la double indexation «**[i][j]**», le premier index renvoie donc à l'index de la ligne, le deuxième index renvoie donc à l'index de la colonne. La commande **len()** renvoie la longueur de la liste.

```
>>> len(a)
3
>>> len(a[1])
2
```

## Exercice 1 : Table de multiplication

Ecrire un script permettant de remplir une liste donnant les résultats de la table de multiplication d'un nombre réel passé en argument. Les résultats seront ensuite affichés par relecture de la liste utilisée sous la forme suivante : (exemple pour la table de multiplication de 1.2 )

Vous cherchez la table de multiplication de quel nombre ?

```
1.2 * 0 = 0.0
1.2 * 1 = 1.2
1.2 * 2 = 2.4
1.2 * 3 = 3.6
1.2 * 4 = 4.8
1.2 * 5 = 6.0
1.2 * 6 = 7.2
1.2 * 7 = 8.4
1.2 * 8 = 9.6
1.2 * 9 = 10.8
```

🚦 Vous allez vous apercevoir qu'avec Python, `1.2 * 3` donne `3.5999999999999996` ! C'est complètement normal étant donné que les fractions sont représentées d'une manière approcher par les ordinateurs. Pour afficher plutôt 3.6, vous pouvez utiliser la fonction `round(x,n)` qui prend un nombre `x` en entrée et l'arrondit à `n` décimales.

## Exercice 2 : Calcul de moyennes

Nous allons dans cet exemple écrire un programme permettant à l'utilisateur de calculer la moyenne de classe ainsi que les écarts à la moyenne d'un examen de la ressource R107. Vu le nombre d'étudiants suivant le cours, il est impossible de stocker la note de chaque étudiant dans une variable distincte. Nous avons donc besoin d'une liste. L'ébauche du programme est la suivante :

```
# Demande le nombre d'étudiants à l'utilisateur
nombreEtudiants = int(input("Donnez le nombre d'etudiants : "))
moyenne = 0.0;
# déclaration d'une liste vide qui va contenir autant de notes que
d'étudiants
notes = []
```

Maintenant que la liste est déclarée, nous pouvons la remplir en demandant à l'utilisateur la note de chaque étudiant. Puisque le nombre d'itérations est connu, nous allons utiliser une boucle `for`. Ainsi, à chaque itération, nous pouvons demander une note et la stocker dans la liste. On en profite aussi pour calculer la somme des notes au fur et à mesure qu'on les reçoit. Une fois ceci fait, il est maintenant possible de calculer la moyenne de classe et d'afficher l'écart à la moyenne pour chaque étudiant.

Compléter le code fourni afin de réaliser l'objectif souhaité. Votre programme ne doit pas autoriser la saisie des notes inférieures à 0 ou supérieures à 20 !

Un exemple d'exécution du programme donnera :

```
Donnez le nombre d'etudiants : 4
```

```
Note etudiant 0 : 4.0
```

```
Note etudiant 1 : 4.0
```

```
Note etudiant 2 : 5.0
```

```
Note etudiant 3 : 6.0
```

```
Moyenne de classe : 4.75
```


```
Numéro de l'Etudiant | note | ecart a la moyenne
```

```
0 | 4.0 | -0.75
```

```
1 | 4.0 | -0.75
```

```
2 | 5.0 | 0.25
```

```
3 | 6.0 | 1.25
```

-  Comment vous pouvez faire pour afficher une phrase avec des variables en appelant la fonction `input()` ?

### Exercice 3 : produit scalaire

Écrivez un programme `Scalaire.py` qui calcule le produit scalaire de deux vecteurs d'entier, implémentés au moyen des listes. Votre programme devra utiliser (entre autres) les éléments suivants :

- Déclarer une variable `nMax` représentant la taille maximale des vecteurs (inutile de lui donner une valeur trop élevée... 3 suffit amplement)
- Déclarer deux listes `v1` et `v2` vides.
- Demander à l'utilisateur d'entrer `n`, la taille effective des vecteurs.
- Vérifier que `n` est compris entre 1 et `nMax` et demander à l'utilisateur d'entrer à nouveau une valeur tant que ce n'est pas le cas
- Demander à l'utilisateur les composantes (`v1[0]... v1[n-1]` , `v2[0] ... v2[n-1]`) des vecteurs `v1` et `v2`.
- Calculer le produit scalaire de `v1` et `v2`.
- Afficher le résultat.

Un exemple d'exécution du programme donnera :

```
Quelle est la taille de vos vecteurs [entre 1 et 10] ? 3
```

```
Saisie du premier vecteur :
```

```
v1[0] = 2
```

```
v1[1] = 3
```

```
v1[2] = 4
```


```
Saisie du second vecteur :
```

```
v2[0] = 1
```

```
v2[1] = 3
```

```
v2[2] = 7
```

Le produit scalaire de  $v_1$  par  $v_2$  vaut 39.0

 **Rappel :** Le produit scalaire de  $a$  par  $b$  noté  $a \cdot b$  est :  $a_1 \cdot b_1 + a_2 \cdot b_2 + \dots + a_n \cdot b_n$   
Exemple : si  $a = (5, 3, -1)$  et  $b = (2, 1, 2)$  alors  $a \cdot b = 11$

### Point d'étape 1 à faire valider

## Exercice 4 : Élément le plus fréquent dans une liste

Le but de cet exercice est d'écrire un programme permettant d'identifier l'élément apparaissant le plus fréquemment dans une liste d'entiers. Ce programme devra également afficher le nombre d'occurrences dans la liste de cet élément le plus fréquent. Par exemple, pour la liste suivante :  $\{2, 7, 5, 6, 7, 1, 6, 2, 1, 7\}$  votre programme devra indiquer que l'élément le plus fréquent est le 7 et que sa fréquence d'apparition est 3.

### Partie 1 :

Récupérer le programme `MostFrequent.py` présent sur moodle que vous complétez afin d'avoir le résultat souhaité. Un exemple d'exécution de votre programme devra produire un affichage se conformant strictement à l'exemple suivant :

```
Le nombre le plus frequent dans la liste est le : 7 (3 x)
```

Le code que vous écrirez devra pouvoir s'appliquer à n'importe quelle liste et vous pourrez supposer que ces listes sont toujours non vides.

Notez à ce propos que si dans une liste donnée il y a plus d'un nombre ayant le plus grand nombre d'occurrences, alors votre programme ne retiendra que celui qui apparaît en premier dans la liste.

Par exemple, pour la liste  $L1 = \{2, 7, 5, 6, 7, 1, 6, 2, 1, 7, 6\}$ , où 6 et 7 sont tous deux les nombres les plus fréquents (les deux apparaissant 3 fois), votre programme ne retiendra que le 7 et affichera :

```
Le nombre le plus frequent dans la liste est le : 7 (3 x)
```

### Partie 2 :

Vous allez à présent modifier votre code de telle sorte à utiliser la méthode `count()` propre aux listes, disponible en Python. Vous allez voir, ça va simplifier pas mal votre code !

## Exercice 5 : Date (exercice vu en TD)

Soit une chaîne de caractères décrivant une date sous la forme `jjmmaaaa`. Cette chaîne peut être vue comme une liste de 8 éléments. Ecrire le programme `VerificationDate.py` qui détermine si la date est valide ou non en affichant un message. Il faut bien détailler tous les cas possibles ! Au cas où l'utilisateur donne une date qui n'est pas au bon format, alors un message doit être affiché en l'invitant à corriger sa saisie.

A noter qu'une année n'est bissextile (elle aura 366 jours) que dans l'un des deux cas suivants :

- Si l'année est divisible par 4 et non divisible par 100 ;
- Si l'année est divisible par 400 (« divisible » signifie que la division donne un nombre entier, sans reste).

Ainsi, si l'année est bissextile alors février aura 29 jours. Sinon l'année n'est pas bissextile : elle a la durée habituelle de 365 jours (elle est dite année commune).

Vérifier votre programme avec les dates suivantes :

- 3102199
- 31041000
- 32052020
- 30032021

## Exercice 6 : Tri de N nombres

Soit une liste de N nombres à trier dans l'ordre croissant. Pour réaliser ce tri, on parcourt le tableau depuis le premier élément jusqu'au dernier de la façon suivante : on prend le premier élément, et on cherche dans le reste du tableau le plus petit nombre lui étant inférieur : si oui on permute les deux éléments. Ensuite, on passe au second élément en réitérant l'opération (jusqu'à avoir traité tous les éléments du tableau).

Exemple de traitement :

Phase 0	<b>5</b>	2	4	8	1	3
Phase 1	1	2	4	8	<b>5</b>	3
Phase 2	1	<b>2</b>	4	8	5	3
Phase 3	1	2	<b>3</b>	8	5	4
Phase 4	1	2	3	<b>4</b>	5	8
Phase 5	1	2	3	4	<b>5</b>	8

- 1) Écrire un programme python permettant de réaliser ce tri. Déclarer une liste avec les éléments de l'exemple ci-dessus. Votre programme doit permettre l'affichage de l'évolution du tri comme donner dans l'exemple suivant :

```
[5, 2, 4, 8, 1, 3]
[1, 2, 4, 8, 5, 3]
[1, 2, 4, 8, 5, 3]
[1, 2, 3, 8, 5, 4]
[1, 2, 3, 4, 5, 8]
[1, 2, 3, 4, 5, 8]
```

- 2) Exécuter le bout de code suivante : `print(sorted(tab))` avec `tab` c'est la liste que vous avez déclarée. Afficher à nouveau la liste `tab`, qu'est-ce que vous remarquez ?
- 3) Faire cette fois-ci `tab.sort()` puis afficher la liste `tab`, qu'est-ce que vous remarquez ?

**Point d'étape 2 à faire valider**

## Exercice 7 : les tuplets

Python offre d'autres types de données pour représenter des tableaux de valeurs. Les «tuplet» en font partie. Comme pour les listes, il s'agit d'une collection de valeurs séparées par des virgules mais cette fois encadrées par des parenthèses.

Exemple : `t=('un', 1, 'deux', 2)`

- Ecrire le programme « `tp4exo7.py` » qui enregistre dans un « tuple » nommé « binome » votre login et celui de votre voisin de tp. Vous afficherez les deux membres du binôme dans une chaîne de caractères du type : « **L'étudiant login1 est en binome avec l'étudiant login2** » où « **login1** » et « **login2** » sont respectivement les valeurs de chaque membre enregistré dans le tuple « **binome** ».
- Vous souhaitez changer à présent de binôme, proposez la saisie du nouveau login et modifier le second élément du tuple « **binome** » en conséquence. Que constatez-vous ?
- Essayez de supprimer le second élément de binôme à présent à l'aide de la fonction intégrée « **del()** » Que constatez-vous ?
- Vous souhaitez former un trinôme en ajoutant un troisième élément au « tuple binome ». Comment procéder ? Que pouvez-vous en conclure sur les « tuple » et leurs domaines d'utilisation ?

## Exercice 8 : Les dictionnaires

Python dispose d'un quatrième type de structure de données après les «**chaînes de caractères**», les «**list**» et les «**tuplet**». Comme les listes, ils sont modifiables, mais ce ne sont pas des séquences car les éléments qu'ils contiennent ne sont pas disposés dans un ordre immuable. Pour accéder à un élément, on utilise une «**clé**», alphabétique, numérique ou même d'un type composite sous certaines conditions. Comme dans une liste, les éléments mémorisés dans un dictionnaire peuvent être de n'importe quel type, valeurs numériques, chaînes, listes, tuples, dictionnaires, et même des fonctions, des classes ou des instances. Ils apparaissent sous la forme d'une série d'éléments séparés par des virgules, le tout enfermé entre deux accolades. Chaque élément est lui-même constitué d'une paire d'objets : **une clé et une valeur**, séparées par un double point. Voici un exemple :

```
>>> dico = {}
>>> dico['computer'] = 'ordinateur'
>>> dico['mouse'] = 'souris'
>>> dico['keyboard'] = 'clavier'
>>> print(dico)
{'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

La première ligne déclare un dictionnaire vide. On ajoute ensuite les éléments en créant simplement une clé d'entrée dans le dictionnaire puis en lui affectant une valeur : on crée les paires clé/valeur.

L'ordre n'a pas d'importance, pas de référencement par rapport à l'ordre de saisie, seule la clé compte. L'accès à un élément particulier s'effectue donc à partir de sa clé (similaire aux listes avec l'index) et l'on peut utiliser les fonctions intégrées du langage Python comme «**len()**», «**del()**»... sur un dictionnaire.

Les dictionnaires sont des objets, ils disposent de fonctions propres permettant de manipuler leurs éléments. Les plus couramment utilisées sont «**dict.keys()**» qui renvoie la séquence des clés du dictionnaire, «**dict.values()**» renvoie la séquence des valeurs mémorisées, «**dict.items()**» extrait du dictionnaire une séquence équivalente de «**tuplet**» représentant les couples «**clé: valeur**»

- Maintenant vous allez écrire le script « `tp4exo8.py` » qui enregistre dans un « dictionnaire » votre prénom avec la clé « `firstname` », votre nom avec la clé « `name` », votre promotion avec la clé « `promo` » et votre groupe de tp avec la clé « `group` ». Vous afficherez ensuite l'ensemble des données vous caractérisant pour chaque couple clé/valeur.

Exemple d'affichage :

vous nom est 'name', prénom est 'firstname', vous faites partie de la promo 'promo' et votre groupe est 'group'

- À présent vous allez créer un dictionnaire nommé « `tpx` » avec x votre numéro de groupe de tp. Ce dictionnaire devra contenir comme clé les identifiants de binôme de tp (votre numéro de poste) et comme valeur les tuplets formés par les dictionnaires représentant eux-mêmes les étudiants formant le binôme. Afficher ligne par ligne les tuplets représentant les membres d'un binôme.

### ***Point d'étape 3 à faire valider***

Référence :

TP de J.Sam, EPFL, TD de F.Drouhin et A. Albert, IUT de Colmar