# TP02 Approfondissement du langage: les listes

Comme ce sera le cas durant toute l'année, votre dossier de travail est accessible[1] sur votre session dans `Mes Documents/Devoirs/fleck/TP02/`.

Les listes sont des outils précieux en programmation. On a souvent à manipuler des ensembles ordonnés d'objets pour en tirer certaines informations comme le nombre ou la valeur du plus grand, etc. Parfois, il est nécessaire de stocker toute une série de valeurs dans un coin pour éviter d'avoir à les recalculer plus tard. Ou tout simplement, on dispose de données temporelles (évolution d'une mesure au cours du temps) qu'il faut bien pouvoir stocker en tant que telle. On a déjà vu comment créer une liste vide (`L = []`) puis comment la remplir « à la main » avec des objets potentiellement stockés dans d'autres variables (`L.append(autre_variable)`). Il reste maintenant à savoir comment accéder aux éléments stockés dans la liste.

Partie I

## Accès à un élément donné via sa position

À chaque élément d'une liste est associé un numéro, un peu comme chaque maison d'une rue a un numéro qui permet au facteur de s'y retrouver. Et de la même manière que pour les maisons, cette numérotation suit certaines règles:

- La numérotation commence toujours à `0`.
- S'il y a `n` éléments, suivant la règle précédente, le dernier élément est numéroté `n-1` (on passe par autant d'étapes quand on compte de `1` à `n` que lorsqu'on compte de `0` à `n-1`).

Il ne vous viendrait pas à l'idée de confondre le numéro d'une maison dans une rue avec les habitants de la maison elle-même (ils peuvent changer au cours des déménagements successifs). De même, il ne faudra pas confondre les *éléments* d'une liste avec les *positions* de ces éléments dans la liste, même s'il est tout à fait possible que les éléments soient des entiers tout comme les positions[2].

Pour accéder à un élément dans une liste, il suffit de donner son numéro à la liste entres crochets (et non entre parenthèses: une liste n'est pas une fonction[3]). Ainsi, si `L` est une liste, alors `L[3]` est le quatrième élément de cette liste.

1. Une liste `L` a été définie dans le module fourni. Stocker dans la variable `quinzieme_element_de_L` le quinzième élément de la liste `L`

Pour parcourir tous les éléments d'une liste, il est souvent utile de disposer d'un compteur qui va de `0` jusqu'à `n-1` où `n` est la taille totale de la liste. Pour accéder à cette dernière information, vous disposez de la fonction `len`, ce qui permet d'écrire un programme du type du suivant qui construit une liste `L2` constituée des carrés des éléments d'une liste `L1` donnée.

```
n = len(L1)             # On récupère la taille de la liste L1
L2 = [0]*n              # On initialise la liste L2 avec des 0
for i in range(n):      # Pour chaque position de la liste L1
    L2[i] = L1[i]**2    # on remplace le 0 de L2 par l'élément de L1 au carré
```

2. Stockez dans la variable `liste_des_cubes` la liste qui contient les cubes des éléments de la liste `L` de l'exercice précédent.

[1] En cas de problème informatique, il est toujours possible de récupérer ce dossier directement sur le site internet `http://pcsi.kleber.free.fr/IPT/`

[2] Une position dans une liste est toujours donnée par un entier, mais l'élément à cette position peut être n'importe quoi: un entier, un flottant, une chaîne de caractère, une autre liste, etc.

[3] « Regardez les objets que vous manipulez ! », comme vous pourrez souvent l'entendre en maths cette année.

3. Les stocks des ingrédients nécessaires à la réalisation d'un onguent très utiles commencent à se vider et les savants vous chargent d'aller en ville acheter une certaine quantité de chaque ingrédient, afin de pouvoir continuer la production pendant le prochain mois.

   Le comptable étant particulièrement pointilleux, il vous donnera exactement la quantité d'argent dont vous avez besoin, pas une pièce de plus. Heureusement vous savez à l'avance le prix de chaque ingrédient et la quantité dont vous avez besoin.

   Écrivez une fonction `prix_total` qui prend en argument deux listes (`prix_au_kg` et `masse_voulue`) qui représentent respectivement les prix au kilogramme de chaque ingrédient et la masse (en kg) nécessaire à la fabrication de l'onguent. Il doit renvoyer la quantité totale d'argent à demander au comptable.

Remarque: quand une liste `L` contient des autres listes, alors les éléments `L[i]` (pour `i` dans `range(len(L))`) sont eux-aussi des listes, donc peuvent accepter la syntaxe des crochets de telle sorte que `L[i][j]` (pour `j` dans `range(len(L[i]))`) est le $j^e$ élément de la $i^e$ liste[4].

4. Un carré magique est une grille carrée (liste de listes) dans laquelle des nombres sont placés de telle sorte que la somme des nombres de chaque colonne, chaque ligne et de chacune des deux diagonales soit la même. De plus, le carré doit contenir une fois chaque nombre, de 1 au nombre de cases de la grille.

   Écrivez une fonction `est_carre_magique` qui vérifie si la grille de nombres fournie en argument est un carré magique. On vous donne aussi le nombre magique `N` auquelles les sommes doivent être égales. On vous assure que tous les nombres sont différents, il n'est donc pas nécessaire de le vérifier. Votre fonction doit renvoyer une liste[5] de trois booléens (`True` ou `False`) qui représentent respectivement les propriétés « Toutes les lignes ont leur somme qui vaut `N` », « Toutes les colonnes ont leur somme qui vaut `N` » et « Toutes les diagonales ont leur somme qui vaut `N` ».

Partie II

## Méthode pop: récupération et effacement d'un élément

On a déjà vu dans le TP précédent comment ajouter un élément en queue d'une liste à l'aide de la méthode `append`. Il est aussi possible d'enlever un élément grâce à la méthode `pop` qui prend en argument la position de l'élément à supprimer. Si aucun argument n'est fourni, `pop` supprimera le dernier élément. Il est à noter que `pop` renvoie l'élément supprimé, ce qui permet de l'utiliser pour faire quelque-chose d'autre dans le programme. Par exemple, dans la console,

```
>>> L = [2,5,8,42,13]  # On définit la liste L
>>> x = L.pop()        # On lui enlève le dernier élément (13) stocké dans x
>>> y = L.pop(0)       # On lui enlève le premier élément (2) stocké dans y
>>> L,x,y              # Vérification
([5, 8, 42], 13, 2)
```

5. Écrivez une fonction `rabotage` qui prend une liste en argument, ne renvoie rien, mais modifie la liste donnée en argument de sorte à supprimer les sixième, cinquième et premier éléments, tout en remettant le cinquième élément supprimé précédemment tout à la fin de la liste.

[4] Les numérotations commençant bien sûr à 0.

[5] La grille sera un carré magique si et seulement si le triplet renvoyé est `[True,True,True]`

Partie III

## Slicing

Dernière chose à savoir concernant les listes avant de pouvoir vous lâcher dans la jungle des exercices: le slicing. C'est en fait une notation plutôt compacte qui permet de copier tout ou partie d'une liste vers une autre liste. Par exemple la notation `L[2:10]` va renvoyer une liste qui contient les éléments de la liste `L` depuis son troisième élément (numéroté 2) jusqu'à son dixième élément (numéroté 9). En effet, par convention, quand Python reçoit une séquence `2:10`, le dernier élément (10) est exclu de la séquence qui va donc de 2 à 9.

6. Stocker dans la variable `liste_slicee_01` la copie de la liste `liste_a_slicer` comprenant tous ses éléments sauf de le premier.
7. Stocker dans la variable `liste_slicee_02` la copie de la liste `liste_a_slicer` comprenant tous ses éléments sauf les 10 derniers.

NB: la « bonne » manière de vraiment *copier* une liste est d'utiliser la construction `L[:]` qui est un raccourci pour `L[0:len(L)]`. En effet, Python fait ce qu'on appelle du passage de variable par référence pour les objets modifiables comme les listes: quand on assigne une liste à une variable, on ne copie pas effectivement la liste dans la variable, mais l'*adresse* de la liste dans la mémoire, ce qui peut faire que deux variables a priori différentes peuvent pointer vers la *même* adresse dans la mémoire, donc modifier l'une peut directement modifier l'autre. Voyez plutôt ce que cela donne dans la console:

```
>>> L1 = [12,24,42]    # Définition de la première liste
>>> L2 = L1            # L2 pointe vers la même case mémoire
>>> L3 = L1[:]         # mais L3 pointe vers une *copie* de L1
>>> L2[1]= 244         # Modifier la liste L2 va donc...
>>> L1,L2,L3           # ...modifier aussi L1, mais pas L3 (qui a sa propre copie)
([12, 244, 42], [12, 244, 42], [12, 24, 42])
```

Partie IV

## `remove`, `min`, `max`, `index`

La méthode `remove` permet de supprimer la première occurrence d'un élément donné dans la liste (quand on est sûr qu'il y est caché). La méthode `index` quant à elle permet de trouver la position (l'indice) du premier emplacement où se trouve un élément donné. Par exemple

```
>>> L = ['hokus','pokus','fidibus']
>>> L.index('pokus')
1
>>> L.remove('hokus')
>>> L
['pokus', 'fidibus']
```

Contrairement aux *méthodes* (`pop`, `index`, `append`) que l'on a vu jusqu'à présent est qui s'appliquaient directement sur la variable contenant la liste (`L.pop()`, `L.index(bidule)` ou `L.append(truc)`), il existe certaines *fonctions* prédéfinies qui peuvent prendre des listes comme argument, comme `min`, `max` ou `sum` dont les effets sont assez évidents:

```
>>> L = [2,5,9,3]  # Définition de la liste
>>> min(L)         # On récupère le plus petit élément de L,
2
>>> max(L)         # le plus grand élément de L
9
>>> sum(L)         # ou la somme de tous les éléments de L.
19
```

L'exercice qui suit[6] fait usage de ces diverses fonctions et méthodes:

8. En parallèle du grand marché de la ville, auquel vous accompagnez vos amis marchands, un ensemble de jeux sont organisés pour les habitants, en particulier la fameuse « course à 3 jambes » : cette course se déroule par équipes de deux personnes dont deux des jambes sont attachées par une corde.

   Afin de constituer les équipes au hasard, une sorte de tirage au sort est organisé mais cela prend beaucoup de temps à faire manuellement, vous décidez d'aider les organisateurs en écrivant une fonction `appariement` qui prend en argument une liste d'entiers différents que chaque participant a librement choisi.

   Les équipes sont constituées ainsi : la personne ayant choisi le plus petit entier est avec celle ayant choisi le plus grand, celle ayant choisi le deuxième plus petit est avec celle ayant choisi le deuxième plus grand, et ainsi de suite.

   Vous devrez renvoyer la liste des compositions de chacune des équipes (chaque personne est identifiée de manière unique par la position de son vote dans la liste donnée en argument), dans l'ordre : d'abord celle dont le plus petit numéro fait partie, puis celle dont le second plus petit numéro fait partie, et ainsi de suite. Au sein de chaque équipe on affichera d'abord le plus petit numéro puis le plus grand. On vous garantit que tous les numéros sont différents. Voici un exemple

   ```
   >>> tirage = [10, 32, 29, 45, 72, 2]
   >>> appariement(tirage)
   [[5, 4], [0, 3], [2, 1]]
   ```

Partie V

## Tableau de booléen annexe pour détecter les tricheurs

Hint: un indice est caché dans le titre de la section

9. Vous êtes employé dans un cinéma et votre patron décide de lancer une offre spéciale. Toute personne possédant une carte de fidélité a le droit, pendant un mois, de voir un film gratuit par jour. Bien entendu certaines personnes vont essayer de tricher en venant plusieurs fois au cinéma dans la même journée et votre travail consiste à détecter ces tricheurs. Si vous trouvez un tricheur, vous devez laisser votre caisse à un collègue, et emmener le tricheur chez votre patron qui lui confisquera sa carte.

   Votre fonction `detection_tricheur` prend en entrée la liste des numéros de carte utilisés dans la journée et doit renvoyer le numéro du premier tricheur (s'il y en a au moins un) ou `None` (sinon). On garantit que les numéros sont tous des entiers positifs « pas trop grands » (de sorte à pouvoir facilement être utilisés comme indices d'une liste).

[6] Ce n'est pas le plus simple du TP, donc si vous coincez, n'hésitez pas à poser des questions ou passer à la suite.

Partie VI

## Gestion de stock

10. Gérard, votre ami gérant d'un supermarché, s'est débarrassé des vieilles caisses enregistreuses et dispose maintenant de tout un système moderne, avec lecteurs de code-barres. Un passage rapide d'un produit devant le lecteur et le nom du produit s'affiche instantanément à l'écran à côté de son prix. Votre ami souhaite utiliser ce système pour maintenir un état complet de son stock de produits et préparer ses commandes en évitant d'avoir à faire l'inventaire toutes les semaines. Lors de chaque achat ou vente d'un produit, l'opération est stockée dans un fichier, accompagnée du numéro du produit. Vous devez écrire un programme (`etat_du_stock`) qui analyse le contenu de ce fichier et détermine la quantité restante de chacun des produits du magasin.

    Les données du fichier sont transmises à votre programme sous forme de deux listes:
    - le nombre de produits de chaque type disponibles dans le magasin, dans l'ordre du type, *avant* que les achats et ventes décrits dans le fichier n'aient été effectués.
    - La liste des opérations effectuées sous forme de liste de listes de deux entiers : le numéro du type de produit qui a été acheté ou vendu, et la quantité de produits concernée. Cette quantité est un entier positif lorsqu'il s'agit d'un achat par Gérard, et négatif lorsqu'il s'agit d'une vente.

    Votre programme doit renvoyer une liste contenant le nombre de produits de chaque type disponible dans le magasin, dans l'ordre du type, *après* que les achats et ventes décrits dans le fichier aient été effectués.

Partie VII

## Append et pop(0): gestion de stock à l'aide d'une file

11. Gérard est fatigué d'avoir à réordonner certaines piles de produits périssables assez rapidement et a donc décidé d'investir dans un système de distribution plus efficace. Avec ce système, dans lequel il place ses produits, les clients se servent automatiquement en bas de la pile et Gérard peut insérer les nouveaux produits tout en haut. Les clients prennent donc les produits dans l'ordre où Gérard les a placés. Ce système réduit les chances qu'il reste des produits périmés, mais ne les supprime pas totalement. Il faut de temps en temps jeter quelques produits lorsqu'il est trop tard pour les vendre. Gérard vous fournit la liste des opérations effectuées et vous demande d'écrire un programme capable de détecter la date d'expiration la plus ancienne parmi les produits restants.

    La liste reçue en paramètre est une liste de doublets où
    - Le premier entier est la quantité de produits concernés par l'opération. Cette quantité est un entier positif lorsqu'il s'agit d'un achat par Gérard (ajout) et négatif lorsqu'il s'agit d'une vente (retrait).
    - Le deuxième entier vaut 0 si l'opération est une vente (retrait). S'il s'agit d'un achat, cet entier représente la date de péremption du produit. L'entier correspond à la concaténation de l'année sur quatre chiffres, du mois sur deux chiffres et du jour sur deux chiffres.

    EXEMPLE entrée :

```
operations = [( 3, 20040810),
              (-1, 0),
              (-1, 0),
              ( 4, 20040920),
              (-1, 0),
              ( 3, 20040916),
              (-3, 0),
              (-2, 0)]
```

    sortie attendue: 20040916