TP02 Approfondissement : listes et dictionnaires

Pensez à utiliser GithubDesktop pour faire un « Pull » et récupérer le dossier du TP qui aura été déposé sur Gitlab par le Général Kléber.

Pensez aussi à écrire votre code *en premier lieu* sur feuille pour vous entraîner aux futures épreuves écrites d'informatique.

Partie I

Listes et dictionnaires : à quoi ça sert ?

Les listes et les dictionnaires sont des outils précieux en programmation car ils permettent de stocker, chacun à sa manière, des données de façon organisée et facilement accessible. Pour les liste, l'accès se fait en fonction de la position dans la liste alors que pour les dictionnaire, on y accède via une « clef » qui permet de retrouver rapidement la position réelle de l'objet stocké [1].

En python, les listes sont signalées entre crochets lors de leur définition et peuvent contenir n'importe quel type d'objet (ici un complexe, un entier et une chaîne de caractères). Vous pouvez remarquer au passage que le premier élément est en position 0 et non 1. Attention à bien le garder en tête.

```
>>> L = [1j, 42, 'bonjour']
>>> L
[1j, 42, 'bonjour']
>>> L[1]
42
```

Remarquez que les crochets sont aussi utilisés pour accéder à un élément de la liste à condition de donner entre crochet la position dans la liste (le premier élément étant en position 0, le deuxième en position 1, ..., le n^e en position $n-1$). Les crochets d'accession (notation partagée avec beaucoup d'objets comme les chaînes de caractères, les dictionnaires, les `np.array`, etc.) ne doivent pas être confondus avec ceux qui permettent de définir une liste, ce doit être deux concepts bien séparés dans votre esprit.

Les dictionnaires, eux, sont définis par des accolades et on met dans l'ordre la clé et la valeur correspondante séparés par un double point. On accède aussi à un élément avec cette notation entre crochet, mais en donnant cette fois la clef d'accès en argument.

```
>>> dico = {'nom': 'Fleck', 'age': 42}
>>> dico
{'nom': 'Fleck', 'age': 42}
>>> dico['nom']
Fleck
>>> dico['age']
42
```

Pour définir une liste vide, on peut mettre des crochets sans rien dedans (`L = []`), de même pour définir un dictionnaire vide avec des accolades (`dico = {}`). Pour un dictionnaire, on peut définir une valeur associée à une clef sans qu'il n'y ait besoin que la clef existe déjà.

```
>>> dico = {}
>>> dico['salutation'] = 'Bonjour'
```

1. Vous verrez en deuxième année une partie du mécanisme astucieux qui permet ce tour de force.

```
>>> dico
{'salutation': 'Bonjour'}
```

En revanche, une liste se plaindra fortement si vous essayez d'assigner une valeur à un emplacement qui n'est pas encore occupé, c'est pourquoi il faut alimenter une liste vide à l'aide de la méthode `append` qui rajoute un élément à la fin de la liste, permettant de construire la liste au fur et à mesure ou alors il faut initialiser tout de suite la liste (par exemple avec des 0) à la bonne taille.

```
>>> L = []
>>> L[0] = 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> L = [0] * 10
>>> L[9] = 42
>>> L
[0, 0, 0, 0, 0, 0, 0, 0, 0, 42]
```

Partie II

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`

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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.

Pour les dictionnaires, le principe est le même sauf qu'on a besoin de connaître les clefs qui permettent d'accéder aux valeurs via la structure `dico[clef]` où `clef` doit être une clef valide du dictionnaire `dico` quand on demande l'accès à la valeur [4].

2. Un dictionnaire `notes_sur_5` a été défini dans le module fourni dont les clefs sont `'QDC01'`, `'QDC02'`, ...jusqu'à `'QDC99'`. Il contient, comme son nom l'indique, les notes d'un élèves pour les 99 premières QDC (Questions De Cours) de l'année. Stockez dans la variable `note_QDC04` la note obtenu pour la quatrième question de cours.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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
```

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

Pour un dictionnaire, la seule information du nombre d'éléments stockés (aussi accessible via `len(dico)`) ne suffit pas à connaître les clefs (qui peuvent être n'importe quoi), mais la méthode `keys()` permet de connaître l'ensemble des clés d'un dictionnaire et donc d'itérer dessus

```
clefs = dico.keys()   # On récupère l'ensemble des clés (ne pas oublier les parenthèses,
dico2 = {}            # On initialise le dictionnaire dico2 à un dictionnaire vide
for k in clefs:       # On itère sur chacune de clefs (key en anglais, d'où le k)
    dico2[k] = dico[k] * 2 # On prend le double de chaque valeur (il faut que ce soit l
```

Python offre aussi un raccourci en permettant d'itérer directement sur le dictionnaire via ses clefs

```
for k in dico:        # On itère sur chacune de clefs du dictionnaire
    dico2[k] = dico[k] * 2 # On prend le double de chaque valeur (il faut que ce soit l
```

4. Stockez dans la variable `notes_sur_20` le dictionnaire avec les mêmes clefs que `notes_sur_5` qui vous est fourni, mais où chaque note a été multipliée par le ce qu'il faut pour faire une note sur 20.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

4. mais pas si on veut l'assigner, ce qui est un avantage des dictionnaires sur les listes.

Partie III

Exercice d'application directe

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.

5. 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, rangés bien sûr dans le même ordre pour qu'ils se correspondent. Elle doit renvoyer la quantité totale d'argent à demander au comptable.
6. On veut résoudre le même problème, mais cette fois-ci `prix_au_kg` et `masse_voulue` sont des dictionnaires qui n'ont pas forcément le même nombre de clefs (`prix_au_kg` contient l'ensemble des articles proposés à la vente alors que les clefs de `masse_voulue` concernent seulement les articles nécessaires à la fabrication de l'onguent [5]). Complétez la fonction `prix_total_avec_dico` qui permette de renvoyer la quantité totale d'argent à demander au comptable.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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 $\texttt{j}^{\text{e}}$ élément de la $\texttt{i}^{\text{e}}$ liste [6].

7. 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 [7] 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 [8] 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` ».

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

5. On assure néanmoins que tous les articles nécessaires à la fabrication de l'onguent sont bien disponibles à la vente.
6. Les numérotations commençant bien sûr à 0.
7. Les éléments de `grille` sont les lignes du carré magique.
8. La grille sera un carré magique si et seulement si le triplet renvoyé est `[True,True,True]`

Partie IV

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`. Si aucun argument n'est fourni, `pop` supprimera le dernier élément avec un coût constant. 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()        # On lui enlève le nouveau dernier élément (42) stocké dans y
>>> L,x,y              # Vérification
([2, 5, 8], 13, 42)
```

8. Écrivez une fonction `inversion` qui prend une liste en argument et renvoie la liste à l'envers (après avoir malheureusement détruit la liste donnée en argument) en utilisant séquentiellement les méthodes `pop` et `append`.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie V

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`.

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

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

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 VI

Dictionnaire annexe pour détecter les tricheurs

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

Hint (bis) : des entiers peuvent aussi être des clefs valides pour un dictionnaire. En fait, tout objet non mutable est une clef valide.

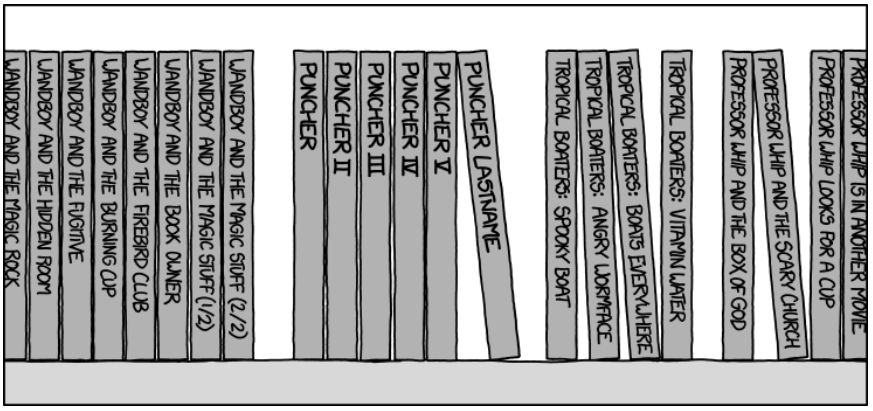
Hint (tris) : pour savoir si une certaine clef `k` est contenue dans un dictionnaire `dico`, on peut utiliser la construction `if k in dico:`.

11. 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 des personnes qui vont venir dans la journée (dans leur ordre d'arrivée) et doit renvoyer le numéro du premier tricheur qui revient voir son film dans la même journée (s'il y en a au moins un) ou `None` (sinon). NB : il peut y avoir plusieurs tentatives de triche, mais seul le premier à repasser en caisse se fait avoir.

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.



There's also the TV show based on the hit Hot and Cold Music books: Fun With Chairs, Royal Rumble, Knife Blizzard, Breakfast for Birds, and Samba Serpents.

xkcd.com

Partie VII

Gestion de stock

12. 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 nom (unique) 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 d'un dictionnaire et d'une liste :

    — le dictionnaire contient le nombre de produits de chaque type disponibles dans le magasin, avec le nom (unique) du produit comme clef, *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 élément : le nom du 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 le dictionnaire contenant le nombre de produits de chaque type disponible dans le magasin *après* que les achats et ventes décrits dans le fichier aient été effectués.

    ATTENTION : il ne faut modifier aucun des deux objets donnés en argument. Il faudra donc faire une copie du dictionnaire en utilisant `new_stock = stock.copie()` pour renvoyer `new_stock` à la fin et ne pas utiliser de méthode vulnérante sur la liste de listes (donc pas de méthode `pop`).

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

Partie VIII

Pile de conserves

13. Dans le supermarché de Gérard, il y a des piles de boîtes de conserves qu'il alimente chaque matin à partir du stock. Les clients se servent bien sûr en haut de la pile de la quantité voulue et il est parfois nécessaire que Gérard réalimente la pile en cours de journée avec des éléments du stock (dont la date de péremption peut varier) : il le fait par le dessus car il n'a pas le temps de s'amuser à tout retrier.

    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 première action est toujours un achat par Gérard car on commence avec une pile vide.

    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 de sorte que la relation d'ordre soit bien celle attendue.

EXEMPLE entrée :

```
operations = [( 3, 20041010),
              (-1, 0),
              (-1, 0),
              ( 4, 20040920),
              (-1, 0),
              ( 3, 20040916),
              (-3, 0),
              (-2, 0)]
```

sortie attendue : 20040920

Stop Gitlab

Allez sur GithubDesktop pour faire un commit. Choisissez (avec pertinence) le résumé. Pensez, si possible, à appuyer sur le bouton «Push origin» en haut à droite pour mettre à jour sur le web.

I am never going out to buy an air conditioner with my sysadmin again.