Université de Lille 1 SCIENCES ET TECHNOLOGIES

ufr d'IEEA
Formations en
Informatique de
Lille 1

L1 MASS/SESI/PEIP - 2016–2017

Algorithmes et Programmation 1

DS1 - 11/03/2017

durée 2h - documents non autorisés (sauf memento Python3)

Ce devoir contient trois exercices indépendants.
Indiquez sur la copie votre parcours et le numéro de votre groupe.
Sauf mention explicite du contraire, toutes les fonctions que vous réaliserez devront comporter une documentation avec contraintes et exemples d'utilisation.

Exercice 1 *Clés d'anagrammes*

L'alphabet considéré dans cet exercice est l'alphabet des 26 lettres latines bas de casse non accentuées, et on pourra considérer définie la constante
`ALPHABET = 'abcdefghijklmnopqrstuvwxyz'`.
Dans le TP sur les anagrammes vous avez construit un dictionnaire dont les clés sont des chaînes de caractères ne contenant que des lettres de l'alphabet, ces lettres apparaissant dans l'ordre alphabétique. Pour cela vous avez contruit une fonction nommée `sort` associant une telle clé à n'importe quel mot (chaîne ne contenant que des lettres) provenant d'un lexique.

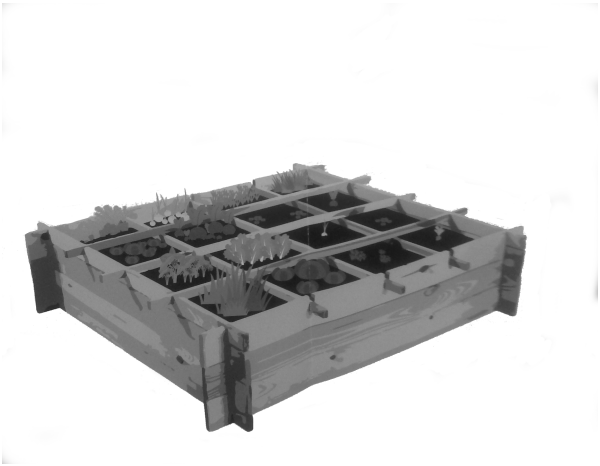
```
>>> sort('timoleon')
'eilmnoot'
```

Question 1 Rappelez comment vous avez programmé la fonction `sort` en utilisant la méthode du même nom sur les listes et la méthode `join` sur les chaînes.

Il est tout à fait possible de réaliser la fonction `sort` sans utiliser la méthode `sort`. Il suffit pour cela de compter le nombre d'occurrences de chacune des lettres de l'alphabet dans le mot, et ensuite construire une nouvelle chaîne en tenant compte de ces nombres d'occurrences.

Question 2 Réalisez une seconde version de la fonction `sort` que vous nommerez `sort2`.

Exercice 2 *Potager*



Dans un potager en carrés, une planche de culture est divisée en carrés de tailles identiques. À chaque carré est affecté un légume. Dans cet exercice une planche de culture est représentée par une liste de listes de mêmes longueurs. Un légume est simplement représenté par son nom sous forme de chaîne de caractère (exemple `"carotte"`). Voici par exemple un potager de 12 carrés :

```
PLANCHE = [
  ['carotte', 'poireau', 'choux', 'navet'],
  ['tomate', 'haricot', 'oignon', 'potiron'],
  ['piment', 'laitue', 'radis', 'aubergine']
]
```

Question 1 Écrire la fonction `carre_valide` prenant en paramètre une planche de culture `p` et un couple d'entiers `c` et qui renvoie `True` si `c` représente les coordonnées d'un carré de la planche `p` et `False` sinon. Avec l'exemple de planche ci-dessus, le couple $(2, 3)$ représente les coordonnées du carré contenant les aubergines, mais le couple $(3, 2)$ ne représente pas un carré valide du potager.

```
>>> carre_valide((2, 3), PLANCHE)
True
>>> carre_valide((3, 2), PLANCHE)
False
```

Question 2 Écrire la fonction `voisins` prenant en paramètre un couple d'entiers `c` et une planche de culture `p` qui renvoie l'ensemble des coordonnées des carrés voisins du carré `c`, c'est-à-dire ceux qui ont un côté en commun. Cette fonction devra renvoyer l'ensemble vide si le carré n'a pas de voisin ou n'a pas de coordonnées valides pour la planche.

```
>>> voisins((2, 3), PLANCHE)
{(1, 3), (2, 2)}
>>> voisins((3, 2), PLANCHE)
set()
>>> voisins((1, 1), PLANCHE)
{(0, 1), (1, 2), (1, 0), (2, 1)}
```

Chaque légume appartient à une *famille botanique* et possède un *type* particulier unique. Deux dictionnaires sont donnés :

— `FAMILLE` : contient des associations de la forme
`'nom_famille' : ensemble des légumes appartenant à cette famille`,
comme par exemple : `"SOLANACEES" : {"aubergine", "tomate", "poivron", "piment"}`

— `TYPE` : contient des associations de la forme
`'nom_type' : ensemble des légumes de ce type`,
comme par exemple : `"RACINE" : {"radis", "navet", "betterave", "carotte"}`

Voici un exemple de chacun de ces deux dictionnaires (incomplet) :

```
FAMILLE = {"APIACEES" : {"carotte", "céleri", "cerfeuil", "persil"},
           "BRASSICACEES" : {"choux", "radis", "navet", "cresson"},
           ...
           "ASTRACEES" : {"laitue", "chicorée"}}

TYPE = {"FEUILLE" : {"blette", "laitue", "chicorée", "céleri", "cerfeuil","persil",
                     "choux","cresson","poireau","ciboulette","épinard"},
        "RACINE"  : {"radis","navet","betterave","carotte"},
        ...
        "BULBE" : {"ail","oignon"}}
```

On suppose que chaque légume présent dans l'un des deux dictionnaires est aussi présent dans l'autre.

Question 3 Définir les variables `NOMS_FAMILLE` et `NOMS_TYPE` contenant sous forme d'ensembles les noms des familles présentes dans le dictionnaire `FAMILLE` et les noms des types présents dans le dictionnaire `TYPE`.

Question 4 Écrire la fonction `legumes` prenant en paramètre un dictionnaire de même structure que les dictionnaires `FAMILLE` et `TYPE` et qui renvoie l'ensemble des légumes présents dans ce dictionnaire.

```
>>> legs =  legumes(FAMILLE)
>>> [leg for leg in legs if leg[0] == 'p']
['piment', 'petit pois', 'poivron', 'persil', 'potiron', 'poireau']
>>> legs =  legumes(TYPE)
>>> [leg for leg in legs if leg[0] == 'a']
['aubergine', 'ail']
```

Question 5 Écrire la fonction `categorie` prenant en paramètre un légume `leg` et un dictionnaire `d` de même structure que les dictionnaires `FAMILLE` et `TYPE` et qui renvoie la catégorie de `leg` selon le dictionnaire `d`, par exemple selon le dictionnaire `TYPE` la catégorie de `"carotte"` est `"RACINE"`. La fonction renvoie `None` dans le cas d'un légume ne figurant dans aucune catégorie du dictionnaire.

```
>>> categorie('carotte', FAMILLE)
'APIACEES'
>>> categorie('carotte', TYPE)
'RACINE'
```

Question 6 Définissez la variable `LEGUME` de type dictionnaire qui contient pour chaque légume l'association `"nom" : ("famille", "type")`, par exemple pour la carotte : `"carotte" : ("APIACEE", "RACINE")`.

Une planche de culture est valide si deux carrés voisins contiennent des légumes de famille et de type différents.

Question 7 Écrire la fonction `planche_valide` prenant en paramètre une planche de culture `p` et renvoyant `True` si aucun légume n'est àvoisin d'un légume de la même famille ou du même type dans cette planche, et `False` sinon.

```
>>> planche_valide(PLANCHE)
False
```

Question 8 Écrire la fonction `proposition` prenant en paramètre une planche de culture `p` et un carré `c` et renvoyant l'ensemble des légumes pouvant être placés dans ce carré en fonction des légumes déjà présents dans les carrés voisins.

Remarque : un carré peut contenir la valeur `None` et la fonction peut renvoyer l'ensemble vide si aucun légume n'est compatible avec le voisinage donné.

```
>>> proposition(PLANCHE, (1,1))
{'petit pois', 'haricot', 'fève', 'carotte', 'betterave', 'radis', 'navet'}
```

Exercice 3 *Records*

Soit l une liste d'entiers de longueur n. Soit $0 \leq k \leq n-1$ un entier. On dit qu'il y a *record vers le bas* au rang k pour la liste l, si l'élément $l[k]$ est strictement inférieur à tous les éléments de la liste dont les indices sont strictement inférieurs à k.

Question 1 Réalisez un prédicat `est_record` paramétré par une liste l et un entier k qui teste s'il y a *record vers le bas*.

```
>>> est_record([31,41,59,26,53,58,97,93,23],0)
True
>>> est_record([31,41,59,26,53,58,97,93,23],1)
False
>>> est_record([31,41,59,26,53,58,97,93,23],3)
True
>>> est_record([31,41,59,26,53,58,97,93,23],8)
True
```

Question 2 En fonction de l'entier k, combien de comparaisons d'éléments de la liste sont effectuées lors de l'appel à `est_record(l,k)` ?

On souhaite maintenant réaliser la liste *records* des indices où il y a record dans la liste l. On se propose d'abord d'utiliser l'algorithme suivant :

Entrée : une liste l
  records $\leftarrow$ liste vide
  $n \leftarrow$ longueur de l
  pour i allant de 0 à $n-1$ **faire**
    si i est un record vers le bas pour l **alors**
      ajouter i à la fin de la liste *records*
    fin si
  fin pour
  renvoyer *records*

Question 3 Réalisez la fonction `liste_record`.

```
>>> liste_record([31,41,59,26,53,58,97,93,23])
[0, 3, 8]
>>> liste_record([31])
[0]
```

Question 4 Déterminez en fonction de la longueur n de la liste l, le nombre de comparaisons effectuées lors de l'appel `liste_record(l)`.

Question 5 Proposez un autre algorithme pour implanter la fonction `liste_record` qui ne fait qu'un seul parcours de la liste.

Question 6 Supposons qu'on trie la liste l en utilisant le tri par insertion vu en cours. Que va-t-il se passer lorsqu'on insère dans la tranche $l[0:k+1]$, l'élément $l[k]$ lorsque il y a record vers le bas ? Proposez une modification du tri par insertion qui tient compte de cette remarque.
