TP (S1) 2 Listes, chaines et dictionnaires

Objectifs

- Savoir utiliser les listes dans des algorithmes simples.
- Savoir effectuer des recherches en utilisant de manière raisonnée les parcours de type `for` ou `while`.
- Savoir parcourir un dictionnaire.

1 LISTES

Exercice 1 Premières manipulations de listes [Sol 1]

1. Si l'on considère deux listes de même longueur n, L $= [\ell_0, \ell_1, .., \ell_{n-1}]$ et M $= [m_0, m_1, .., m_{n-1}]$ d'entiers, , on peut former deux autres listes d'entiers de longueur n également, notées S et P, telles que pour chaque indice $i \in [0, n-1]$, l'élément s_i de la liste S (resp. l'élément p_i de la liste P) soit donné par $s_i = \ell_i + m_i$ (resp $p_i = \ell_i \times m_i$).

Écrire le script de la fonction `som_prod(L:list,M:list)->(list,list)` qui prend en argument deux listes `L` et `M` de même longueur et qui renvoie le tuple `(S,P)` des listes définies ci-dessous.

2. Pour deux ensembles E $= \{e_i\}$ et F $= \{f_j\}$, on appelle produit cartésien l'ensemble des couples $(e_i, f_j), e_i \in$ E, $f_j \in$ F. Écrire le script de la fonction `prod_cart(E:list,F:list)->list` qui prend en argument deux listes `E` et `F` contenant les éléments des ensembles E et F, et renvoyant la liste des éléments du produit cartésien de E avec F, chaque élément du produit étant stocké sous la forme d'une liste à deux éléments. Par exemple, l'appel

```
prod_cart([1,2],['a','b','c'])
```

renverra :

```
[[1, 'a'], [1, 'b'], [1, 'c'], [2, 'a'], [2, 'b'], [2, 'c']]
```

Exercice 2 Autour du maximum d'une liste [Sol 2]

1. On veut écrire la fonction `maximum(L:list)->int` renvoyant la valeur du plus grand élément de la liste `L` (celle-ci étant supposée non vide, et ne contenir que des entiers). On donne l'algorithme sur lequel repose la fonction :

- affecter le premier élément de la liste `L` à une variable `M`,
- pour chacun des éléments `e` de `L`,
  - ◇ si `e` est plus grand que `M`, affecter la valeur de `e` à `M`.

Écrivez cette fonction en utilisant un parcours de la liste à l'aide de l'instruction `for e in L`. Testez la fonction sur des listes présentant des situations diverses (maximum en première position, en dernière position, ailleurs que sur les bords, liste à un seul élément, élément maximal apparaissant plusieurs fois dans la liste).

2. On peut noter que l'emploi de `for e in L` entraîne un test inutile (celui du premier élément de `L`). Cependant la syntaxe associée est très simple et bien adaptée à la situation. Modifier la fonction précédente en utilisant un slicing sur `L` pour remédier à ce problème.

3. Écrivez sur le même principe une fonction `minimum(L:list)->int` renvoyant cette fois la valeur du plus petit élément de la liste `L`, et testez-la sur plusieurs listes bien choisies.

4. Écrivez une fonction `ind_maximum(L:list)->int` renvoyant le plus petit indice `k` tel que `L[k]` soit le plus grand élément de la liste `L`, et utilisant une boucle for. Par exemple, pour la liste `L = [2,4,1,5,3]`, la fonction renverra 3 puisque le plus

grand élément qui vaut 5 apparaît à l'indice 3. Quel type de parcours de liste est-il nécessaire d'utiliser ici ?

5. Réécrire la fonction précédente en utilisant une boucle **while**.

Exercice 3 Présence d'un élément dans une liste [Sol 3]

1. On cherche à écrire une fonction `ind_present(L:list,e)->list` qui renvoie la liste des indices où l'élément `e` apparait dans la liste `L`. Si `e` n'appartient pas à `L`, la fonction devra renvoyer la liste vide.

   1.1) Le parcours de liste doit-il ici se faire à l'aide d'une boucle **for** ou d'une boucle **while** ?

   1.2) Écrire la fonction demandée.

2. On cherche désormais à écrire une fonction `present(L:list,e)->bool` qui renvoie **True** si l'élément `e` appartient à `L`, et **False** sinon. Un élève propose la fonction suivante :

```
def present(L:list,e)->bool:
    """ Renvoie True si 'e' est un élément de 'L', False sinon \
    ↪ """
    pres = False
    for v in L:
        if v == e:
            pres = True
    return pres
```

   Expliquer quel est le défaut de cette façon de programmer. Comment améliorer la fonction précédente ?

3. On donne ci-dessous le code incomplet d'une fonction `present2` qui réalise la même action que `present`, en utilisant cette fois-ci une boucle **while** :

```
def present2(L:list,e)->bool:
    """ Renvoie True si e est un élément de L et False sinon """
    i = 0
    n = len(L)
    present = False
    while i < n and not present:
        if ..... :
            present = True
        i = ......
    return present
```

   3.1) Compléter le code pour rendre la fonction opérationnelle.

   3.2) Expliquer la nécessité de mettre le test `i < n` dans la condition du **while**.

4. Une autre possibilité, sans utiliser le booléen `present`, est de parcourir la liste en utilisant un indice `i` tant qu'on n'est pas au bout de la liste et tant que l'élément courant `L[i]` est différent de l'élément `e` recherché. Proposer le code d'une fonction `present3` qui suit ce principe.

Exercice 4 Doublons dans une liste [Sol 4]

On cherche à écrire une fonction `sansDoub(L:list)->bool` qui prend comme argument une liste et qui renvoie **True** si les éléments de `L` sont tous distincts, et **False** sinon.Pour une liste `L` de taille n, on peut procéder de la manière suivante :

- On compare les valeurs $L[i]$ et $L[j]$ pour tous les couples (i,j) vérifiant $0 \leq i < n-1$ et $i+1 \leq j < n$,
- si on trouve deux éléments égaux, on interrompt la recherche et on renvoie **False**,
- si la recherche se poursuit sur l'ensemble des couples $L[i], L[j]$ sans trouver d'élément commun, on renvoie **True**.

L'algorithme décrit peut être codé à l'aide d'un double parcours réalisé par de deux boucles **for** imbriquées, associé une interruption de boucle imposée par un **return** :

```
def sansDoub(L:list)->bool:
    n = len(L)
    for i in range(n-1):
        for j in range(i+1,n):
            if L[i] == L[j]:
                return False
    return True
```

Cependant, le parcours par boucle **for** n'étant pas forcément mené à terme, il est plus naturel d'utiliser un parcours par boucle **while**.
Écrire le script de la fonction `sansDoub(L:list)->bool` en utilisant des boucles **while** à la place des boucles **for**. On pourra introduire une variable booléenne `noDoub` qui vaut **True** en l'absence de doublons, **False** sinon.

Exercice 5 Tri à bulles [Sol 5]

On s'intéresse ici à une méthode de tri appelée *tri à bulles*, qui s'applique à un ensemble de données contenues dans une liste de n éléments. Le principe de base en est le suivant : on commence par parcourir les éléments de `T` pour un indice i variant

de 0 à $n-2$ (soit $n-1$ valeurs de i), et à chaque itération, si `T[i]>T[i+1]`, alors on permute `T[i]` et `T[i+1]`. L'élément le plus grand est alors en dernière position de `T`. On recommence ensuite le processus en diminuant à chaque fois d'une unité le nombre d'éléments parcourus depuis le début de `T`. Cette méthode est nommée tri à bulles car les éléments les plus grands remontent petit à petit vers la fin de la liste comme des bulles dans une boisson gazeuse.

1. Écrire une fonction `triBulle(T: list) -> list` qui effectue le tri de `T` selon le principe décrit précédemment.
2. Appliquer à la main le principe de ce tri sur la liste `[5, 1, 2, 4, 3]`. Comment l'algorithme pourrait-il être amélioré ?
3. Proposer une nouvelle version de `triBulle(T: list) -> list` qui tienne compte de cette amélioration.

2 CHAINES DE CARACTÈRE

Exercice 6 **Recherche dans un texte** [Sol 6] Le code ACSII (prononcer « ASKI ») est un standard de codage informatique des caractères. Il s'agit d'une table de correspondance entre un caractère et un entier. Ainsi 256 caractères (alphabétiques, numériques, de ponctuation, etc...) sont chacun associés à un entier compris entre **0 et 255**. En outre les lettres majuscules sont codées par des entiers successifs. La commande `chr(i)` renvoie le caractère correspondant à l'entier `i` et la commande `ord(c)` renvoie l'entier associé au caractère c.

1. **1.1)** Déterminer l'entier correspondant au codage du caractère `'A'`.

   1.2) En déduire une fonction `maj(c:str)->bool` qui renvoie `True` si le caractère `c` est une lettre majuscule et `False` sinon
2. Écrire une fonction `compte(s:str,c:str)->int` qui renvoie le nombre d'occurrences du caractère `c` dans la chaîne de caractère `s`.
3. **3.1)** Écrire une fonction `listeMajuscules(s:str)->list` qui utilise la fonction `compte` et qui renvoie la liste des occurrences de chaque lettre majuscule de l'alphabet dans la chaîne de caractère `s`.
   Par exemple, pour la chaine `s = 'aBcDGe'`, l'appel à `listeMajuscule(s)` renverra la liste `[0,1,0,1,0,0,1,0,...,0]` (liste de taille 26).

   3.2) Combien de fois la chaîne de caractère `s` est-elle parcourue lors de l'exécution `listeMajuscules(s)` ?
4. Écrire une fonction `listeMajuscules2(s:str)->list` qui renvoie la liste des occurrences de chaque lettre majuscule dans la chaîne de caractère `s` en **ne parcourant qu'une seule fois** la chaîne de caractère `s`.

Exercice 7 **Autour des palindromes** [Sol 7] Un `palindrome` est un texte qui peut être lu indifféremment de gauche à droite ou de droite à gauche. Par exemple le mot radar est un palindrome.

1. Écrire une fonction `reverse(s:str)->str` qui renvoie la chaîne symétrique de la chaîne de caractère `s`. Par exemple, `reverse('texte à lire')` renvoie `'eril à etxet'`.
2. En utilisant la fonction `reverse`, écrire une fonction `palind1(s:str)->bool` qui renvoie `True` si la chaîne de caractère `s` est un palindrome et `False` sinon.
3. La méthode `lower()` passe en minuscule tous les caractères d'une chaîne. Par exemple, si la chaine `s` contient `textE`, `s.lower()` renvoie la chaine `'texte'`.

   Écrire une fonction `palind2(s:str)->bool` qui renvoie `True` si la chaîne de caractère `s` est un palindrome, `False` sinon, en ayant au préalable transformé les majuscules en minuscules.
4. On désigne par `carExclus` la liste des caractères non alphabétiques.

   Écrire une fonction `palind3(s:str,carExclu:list)->bool` qui renvoie `True` si la chaîne de caractère `s` est un palindrome, `False` sinon, en ayant au préalable transformé les majuscules en minuscules et éliminé tous les caractères non alphabétiques.

3 DICTIONNAIRES

Exercice 8 **Comptage des éléments** [Sol 8] On souhaite écrire une fonction `compte(L : list) -> dict` qui pour une liste `L` donnée renvoie un dictionnaire (notez que `dict` est le nom du type associé la structure de dictionnaire) ayant pour clés les différents éléments de la liste et pour valeur associée le nombre de fois où cet élément est présent dans `L`.

Par exemple, pour la liste `L=[1,5,1,2,5,5,3,2]` où l'élément 1 apparaît 2 fois, l'élément 5 apparaît 3 fois, ..., l'appel `compte(L)` doit renvoyer le dictionnaire `{1:2,5:3,2:2,3:1}`. (Notez que sur cet exemple les clés sont des entiers et pas des chaînes de caractères.)

Par concevoir cette fonction, on propose la démarche suivante :

- on initialise une variable `D` à un dictionnaire vide (qui se note {}, de la même manière que `[]` représente la liste vide) ;
- puis on parcourt chaque élément e de la liste `L` : si e n'est pas déjà présent comme clé dans le dictionnaire `D` (ce qui signifie que c'est la première fois que l'on trouve la valeur e dans la liste `L`) on ajoute cette clé au dictionnaire en lui associant la valeur 1 ; et si au contraire e apparaissait déjà en tant que clé dans le dictionnaire on modifie la valeur associée en lui ajoutant 1 (pour tenir compte de cette nouvelle occurrence de la valeur).

1. Écrire cette fonction et testez-la sur plusieurs listes bien choisies.

2. Modifiez la fonction précédente pour écrire une fonction `indices(L : list) -> dict` renvoyant toujours un dictionnaire dont chaque clé est un élément présent dans `L`, mais la valeur associée est la liste de tous les indices où cet éléments apparaît.

   Par exemple, pour la liste `L = [1, 5, 1, 2, 5, 5, 3, 2]` où l'élément 1 apparaît en indices 0, 2 et l'élément 5 apparaît en indices 1, 4, 5 …, l'appel `indices(L)` doit renvoyer le dictionnaire `{1:[0,2],5:[1,4,5],2:[3,7],3:[6]}`.

3. On considère maintenant le problème inverse de la question précédente. Soit `D` un dictionnaire dont les clés sont les éléments présents dans une liste `L` et dans lequel la valeur associée à chaque clé est la liste des indices où cet élément est présent dans `L`. Écrire la fonction `dict2list(D: dict) -> list` qui reconstruit et renvoie la liste `L` connaissant `D`.

Exercice 9 Recherche de valeur maximale

[Sol 9] On considère un dictionnaire `D` tel que chaque clé de `D` soit une chaine de caractère (de type `str`), et tel que la valeur associée à chaque clé soit un nombre entier. Par exemple, on pourra avoir :

```
D = {'a':5,'b':3,'c':8 }.
```

1. Écrire le script d'une fonction `trouveCle(D:dict,v:int)->list` qui renvoie la liste des clés ayant pour valeur `v` (si aucune clé n'a pour valeur `v`, la fonction renverra la liste vide).

2. Écrire le script d'une fonction `cleMax(D:dict)->str` renvoyant la clé pour laquelle la valeur associée est maximale (dans le cas où plusieurs clés possèdent cette valeur maximale, on renverra indifféremment une de ces clés). On pourra s'inspirer de la méthode vue dans l'exercice 2, et utiliser l'instruction `M = -float('inf')` qui permet de créer une variable `M` plus petite que n'importe quel nombre flottant ou entier.

3. Reprendre la question précédente en écrivant le script d'une fonction `cleMin(D:dict)->str` renvoyant la clé pour laquelle la valeur associée est minimale.

Exercice 10 Mémoization

[Sol 10] Dans cet exercice, on souhaite revenir sur la fonction `sansDoub` d'un précédent exercice. Rappelons que dans cette fonction, on parcourt tous les couples $(L[i], L[j])$ avec $i < j$ et on renvoie **`False`** dès qu'un couple vérifiant $L[i] = L[j]$ est trouvé, **`True`** dans le cas contraire.

1. Il est possible de faire mieux, en créant un dictionnaire `D` qui au début sera initialisé à :

   ```
   D = {x:False for x in L}
   ```

   On parcourt ensuite la liste `L` avec une variable `x`, et :

   - si `D[x] = False`, on passe la valeur à **`True`**,
   - sinon : c'est qu'on avait déjà croisé l'élément `x` et donc la liste `L` contient un doublon !

   L'avantage de cette méthode est qu'elle nous fait économiser beaucoup de parcours de `L` (l'accès à la valeur d'un dictionnaire se faisant quant à lui à « coût constant »). Programmer la fonction `sansDoubMemo(L:list)->bool` selon ce principe. *Le gain est énorme, au second semestre nous pourrons montrer que le temps d'exécution est proportionnel à la longueur n de la liste pour la version avec mémoization, alors que pour la version naïve il est proportionnel à n^2 !*

2. Selon le même principe, proposer une fonction `LsansDoubMemo(L:list)->list` qui renvoie une version sans doublon de la liste `L`. Par exemple, `LsansDoubMemo([2, 3, 3, 1, 2])` renverra `[2, 3, 1]`. Bien sûr, vous noterez qu'il n'y a pas unicité de la réponse, on s'assurera que la fonction renvoie l'une des listes solution.

Exercice 11 Construction d'un index pour un texte

[Sol 11] Soit `c` une chaîne de caractères contenant des mots séparés par un espace (typiquement une phrase), créer une fonction d'en-tête `index(c:str)->dict` qui renvoie un dictionnaires de clefs les mots de `s`, et de valeurs la liste des indices (hors espaces) où ce mot apparaît.
Indication : On pourra utiliser la méthode `split` sur les chaînes comme présenté ci-desous

```
>>> c = "L'ITC c'est génial, non ?"
>>> liste_mots = c.split()
>>> liste_mots
["L'ITC", "c'est", 'génial,', 'non', '?']
```

Tester par exemple sur la chaîne :

```
c = "le temps est beau le ciel est bleu le lycée est beau et \
↪ bleu"
```

SOLUTIONS DES EXERCICES

Solution 1

1. Fonction `som_prod`

```
def sum_prod(L:list,M:list)->(list,list):
    """ renvoie les listes S et P contennant  la somme et le \
    ↪ produit terme à terme des éléments de L et M """
    n = len(L)
    S,P = [],[]
    for i in range(n):
        S.append(L[i]+M[i])
        P.append(L[i]*M[i])
    return (S,P)
```

2. Fonction `prod_cart`

```
def prod_cart(E:list,F:list)->list:
    """ renvoie la listes P contennant les éléments du produit \
    ↪ cartésien de E avec F """
    n = len(E)
    m = len(F)
    P = []
    for i in range(n):
        for j in range(m):
            P.append([E[i],F[j]])
    return P
```

Solution 2

1.

```
def maximum(L:list)->int :
    """ Renvoie le plus grand élément de la liste non vide
    d'entiers L """
    M = L[0]
    for e in L:
        if e > M:
            M = e
    return M
```

2. Avec slicing, les éléments de `L` à tester sont ceux contenus dans `L[1:]`.

```
def maximum(L:list)->int :
    """ Renvoie le plus grand élément de la liste non vide
    d'entiers L """
    M = L[0]
    for e in L[1:]:
        if e > M:
            M = e
    return M
```

3. On remplace > par < (le changement du nom de la variable M en *m* est purement esthétique).

```
def minimum(L:list)->int :
    """ Renvoie le plus petit élément de la liste non vide
    d'entiers L """
    m = L[0]
    for e in L[1:]:
        if e < m:
            m = e
    return m
```

4. Évidemment on boucle ici sur les **indices** :

```
def ind_maximum(L:list)->int :
    """ Renvoie un indice du plus grand élément de la liste \
    ↪ non vide
    d'entiers L """
    ind_M = 0
    for i in range(1,len(L)):
        if L[i] > L[ind_M]:
            ind_M = i
    return ind_M
```

Notez que si le maximum est présent plusieurs fois dans la liste, ce programme renvoie l'indice de sa première occurrence.

5. On peut aussi utiliser une boucle `while` :

```
def ind_maximum(L:list)->int :
    """ Renvoie un indice du plus grand élément de la liste \
    ↪ non vide d'entiers L """
    ind_M = 0
    i = 1
    while i < len(L):
        if L[i] > L[ind_M]:
```

```
            ind_M = i
        i += 1
    return ind_M
```

Solution 3

1. **1.1)** On doit tester les n éléments de la liste `L`, quel que soit son contenu. Il est préférable d'utiliser une boucle **`for`**.

   1.2) Fonction `ind_present`

```
def ind_present(L:list,e)->list :
    """ renvoie la liste des indices où e est présent dans \
    ↪ la liste L """
    L1 = []
    n = len(L)
    for i in range(n):
        if L[i] == e:
            L1.append(i)
    return L1
```

2. La boucle **`for`** parcourt l'intégralité de la liste, même si l'élément recherché est présent au début de la liste. Il vaut donc mieux utiliser une boucle **`while`** qui parcours la liste *tant qu'on a pas trouvé l'élément et qu'on n'est pas au bout de la liste.*

3. **3.1)** La fonction complétée :

```
def present2(L:list,e)->bool:
    """ Renvoie True si e est un élément de L et False \
    ↪ sinon """
    i = 0
    n = len(L)
    present = False
    while i < n and not present:
        if L[i] == e:
            present = True
        i += 1
    return present
```

   3.2) Le test est nécessaire car s'il n'est pas présent et que `e` n'est pas dans la liste `L`, alors on va incrémenter i jusqu'à la valeur n, et le test `L[i] == e` renverra une erreur car il n'y a pas d'élément d'indice n (le dernier élément a pour indice $n-1$).

4. En suivant les indications, on peut proposer :

```
def present3(L:list,e)->bool:
    """ Renvoie True si e est un élément de L et False sinon """
    i = 0
    n = len(L)
    while i < n and L[i] != e:
        i += 1
    return i < n
```

Solution 4 En suivant les indications, on peut proposer :

```
def sansDoub(L:list)->bool:
    noDoub = True # booléen indiquant l'absence de doublon \
    ↪ rencontré
    n = len(L)
    i = 0
    while i < n-1 and noDoub:
        j = i+1
        while j < n and noDoub:
            if L[i] == L[j]:
                noDoub = False
            j += 1
        i += 1
    return noDoub
>>> L = [1, 1, 3, 2]
>>> sansDoub(L)
False
>>> L = [1, 3, 2]
>>> sansDoub(L)
True
```

Solution 5

1. Fonction `triBulle(T:list)->list`

```
def triBulle(T: list) -> list:
    """
    entrée : T liste d'éléments comparables
    sortie : T triée
    """
    n = len(T)
```

```
    for t in range(n):  # n-t-1 est la taille de la liste que \
    ↪ la variable i parcourt
        for i in range(n-t-1):
            if T[i] > T[i+1]:
                T[i], T[i+1] = T[i+1], T[i]
    return T
>>> L = [3, 2, 1]
>>> triBulle(L)
[1, 2, 3]
>>> L # la fonction agit aussi par effets de bord (L a été \
↪ modifiée)
[1, 2, 3]
```

2. Pour la liste donnée, à la fin de la deuxième itération, la liste est déjà triée, et la troisième itération ne conduit à aucune permutation de valeurs (ni la quatrième bien sûr). On peut donc interrompre l'algorithme dès qu'un parcours ne conduit à aucun échange.

3. Nouvelle version de `triBulle(T:list)->list`

```
def triBulle(T:list)->list:
    """
    entrée : T liste d'éléments comparables
    sortie : T triée
    """
    n = len(T)
    t = n-1  # indice du dernier
    echange = True
    while t > 0 and echange:
        echange = False
        for j in range(t):
            if T[j] > T[j+1]:
                T[j], T[j+1] = T[j+1], T[j]
                echange = True
        t -= 1
    return T
>>> L = [3, 2, 1]
>>> triBulle(L)
[1, 2, 3]
>>> L # la fonction agit aussi par effets de bord
[1, 2, 3]
```

Solution 6

1. 1.1)

```
>>> ord('A')
65
```

1.2)

```
def maj(c:str)->bool:
    a = ord(c)
    if a >= 65 and a < 91:
        return True
    else:
        return False
```

2.

```
def compte(s:str, c:str)->int:
    n = 0
    for l in s:
        if l == c:
            n += 1
    return n
```

3. 3.1)

```
def listeMajuscules(s:str)->list:
    res = []
    for i in range(65, 91):
        c = chr(i)
        res = res + [compte(s,c)]
    return res
>>> listeMajuscules("aBcDGe")
[0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
↪ 0, 0, 0, 0, 0, 0, 0]
```

3.2) On se rend compte que la fonction précédente utilise 26 fois la fonction `compte` et parcours donc 26 fois la chaine de caractère.

4.

```
def listeMajuscule2(s:str)->list:
    res = [0]*26
    for c in s:
        if maj(c):
            a = ord(c)
            res[a-65] += 1
    return res
```

Solution 7

1.

```
def reverse(s:str)->str:
    res = ''
    for c in s:
```

```
        res = c+res
    return res
>>> reverse('ete')
'ete'
```

On peut aussi utiliser le slicing :

```
def reverse(s:str)->str:
    return s[::-1]
>>> reverse('ete')
'ete'
```

2.
```
def palind1(s:str)->bool:
    return s == reverse(s)
>>> reverse('ete')
'ete'
>>> reverse('coucou')
'uocuoc'
```

3.
```
def palind2(s:str)->bool:
     sl = s.lower() # On transforme les majuscules
     return sl == reverse(sl)
>>> palind2('etE')
True
>>> palind2('coUcoU')
False
```

4.
```
def palind3(s:str,carExclu:list)->bool:
    N = s.lower() # On transforme les majuscules
    R = '' # On supprime les caractères non alphabétiques, on \
    ↪ crée une nouvelle chaîne "standardisée" dans R
    for c in N:
        if not(c in carExclu):
           R += c
    return R == reverse(R)
>>> palind3("@etE@", ["@", "~"])
True
>>> palind3("coU~co@U", ["@", "~"])
False
```

Solution 8

1. La fonction :

```
def compte(L : list) -> dict :
    """ renvoie un dictionnaire dont les clés sont les éléments
    présents dans la liste L et les valeurs associées sont
    le nombre de fois que cet élément est présent."""
    D = {}
    for e in L:
        if e not in D:
            D[e] = 1
        else :
            D[e] += 1
    return D
>>> L = [1, 5, 1, 2, 5, 5, 3, 2]
>>> compte(L)
{1: 2, 5: 3, 2: 2, 3: 1}
```

2. La fonction :

```
def indices(L : list) -> dict :
    """ renvoie un dictionnaire dont les clés sont les éléments
    présents dans la liste L et les valeurs associées sont la \
    ↪ liste
    des indices où cet élément est présent."""
    D = {}
    for i in range(len(L)):
        if L[i] not in D:
            D[L[i]] = [i]
        else :
            D[L[i]].append(i)
    return D
>>> L = [1, 5, 1, 2, 5, 5, 3, 2]
>>> D = indices(L)
>>> D
{1: [0, 2], 5: [1, 4, 5], 2: [3, 7], 3: [6]}
```

3.
```
def dict2list(D:dict)->list:
    """ renvoie la liste L correspondant au dictionnaire D \
    ↪ obtenu par
    indices(L) """
    # détermination de la taille de L
    n = 0
    for e in D:
        n += len(D[e])
```

```
    # initialisation de L
    L = [0]*n
    # modification des éléments de L
    for e in D:
        for i in D[e]:
            L[i] = e
    return L
>>> dict2list(D)
[1, 5, 1, 2, 5, 5, 3, 2]
```

Solution 9

1.
```
def trouveCle(D:dict,v:int)->list:
    L = []
    for c in D:
        if D[c] == v:
            L.append(c)
    return L
```

2.
```
def cleMax(D:dict)->str:
    M = -float('inf')
    for c in D: # on parcourt les clés
        if D[c] > M:  # cas où la valeur associée à c est plus \
        ↪ grande que le maximum local
            M = D[c]
            cMax = c
    return cMax
```

3. On remplace > par < (le changement du nom de la variable M en *m* est purement esthétique).

```
def cleMin(D:dict)->str:
    m = float('inf')
    for c in D: # on parcourt les clés
        if D[c] < m: # cas où la valeur associée à c est plus \
        ↪ petite que le minimum local
            m = D[c]
            cMin = c
    return cMin
```

Solution 10

1.
```
def sansDoubMemo(L:list)->bool:
    noDoub = True # booléen indiquant l'absence de doublon \
    ↪ rencontré
    D = {x:False for x in L}
    n = len(L)
    i = 0
    while i < n-1 and noDoub:
        x = L[i]
        if not D[x]:
            # élément x rencontré pour la 1ère fois
            D[x] = True
        else:
            # élément x déjà rencontré
            noDoub = False
        i += 1
    return noDoub
>>> L = [1, 1, 3, 2]
>>> sansDoub(L)
False
>>> L = [1, 3, 2]
>>> sansDoub(L)
True
```

2.
```
def LsansDoubMemo(L:list)->list:
    L_sd, D = [], {x:False for x in L}
    for x in L:
        if not D[x]:
            # x n'est pas dans L_sd
            L_sd.append(x)
            D[x] = True
    return L_sd
>>> LsansDoubMemo([2, 3, 3, 1, 2])
[2, 3, 1]
```

Solution 11

```
def index(c):
    liste_mots = c.split()
    indice = 0 # indice du mot dans le texte
    index = {}
    for mot in liste_mots:
```

```
        if mot not in index:
            index[mot] = [indice]
        else:
            index[mot].append(indice)
        indice += len(mot)
    return index
>>> c = "le temps est beau le ciel est bleu le lycée est beau et \
↪ bleu"
>>> index(c)
{'le': [0, 14, 27], 'temps': [2], 'est': [7, 20, 34], 'beau': \
↪ [10, 37], 'ciel': [16], 'bleu': [23, 43], 'lycée': [29], 'et': \
↪ [41]}
```