Algorithmique et Programmation 1

TD5 : Fonctions

1 Pour commencer

1. Écrire une fonction `eurosVersDollars` qui, à partir d'un nombre flottant représentant un montant en €, retourne le montant équivalent en dollars américains. On rappelle que 1€ est équivalent à $1.17.
2. Écrire une fonction `doubler` qui, à partir d'une liste `lst` de nombres (entiers ou flottants) modifie `lst` en multipliant chaque éléments de `lst` par deux.
3. Écrire une fonction `creerDouble` qui, à partir d'une liste `lst` de nombres (entiers ou flottants) ne modifie pas `lst` mais crée une nouvelle liste composée des éléments de `lst` multipliés par 2.
4. Écrire une fonction `indice` qui, étant donnée une liste d'entiers `lst` et un entier `n` retourne l'indice de la première occurrence de `n` dans `lst`. Si `lst` ne contient aucun élément égal à `n` alors cette fonction retournera -1. Par exemple :
   — `indice([2,4,6,8,4],3)` retourne -1 car 3 n'appartient à la liste ;
   — `indice([2,4,6,8,4],4)` retourne 1

   Pour cette question, je vous demande de ne pas utiliser la méthode `index` des listes.

2 Des fonctions qui appellent d'autres fonctions

1. Depuis le module `random`, importer la fonction `random` qui retourne un nombre flottant choisi au hasard dans l'intervalle [0,1[.
2. Écrire une fonction `affichageDes` qui ne prend aucun argument et qui ne retourne aucune valeur. Elle affiche un entier choisi au hasard entre 1 et 6 (inclus).
3. Écrire une fonction `lancerDes` qui ne prend aucun argument et qui n'affiche rien, mais retourne un entier choisi au hasard entre 1 et 6 (inclus).
4. Écrire une fonction `lancer2Des` qui ne prend aucun argument et n'affiche rien. Elle retourne un couple d'entiers choisis au hasard entre 1 et 6. Je vous demande de ne pas utiliser la fonction `random` mais d'utiliser les fonctions que vous avez déjà écrites ci-dessus.
5. Soient l'appel suivant :

   ```
   cpl = lancer2Des()
   ```

   Quel est le type de `cpl` ? Comment peut-on accéder la valeur du premier dé ?

   ```
   (d1,d2) = lancer2Des()
   ```

   Même question pour l'appel ci-dessus.

3 À propos du projet labyrinthe

Le labyrinthe est composé de *cellules*. Chaque cellule valant 0 (mur), 1 (couloir), 2 (entrée) ou 3 (sortie). Chaque cellule est désignée par un numéro de ligne et un numéro de colonne représentant ses coordonnées. On dit que deux cellules sont voisines si la distance entre elles est exactement de 1. Dans un premier temps, imaginons que le labyrinthe est infini (il n'y a pas de bords).

1. Quel est le nombre de voisins de chaque cellule ? (Inutile d'écrire une fonction).
2. Écrire une fonction `afficheVoisins_laby_inf` qui, à partir des coordonnées entières lgn et col d'une cellule, affiche les coordonnées des cellules voisines de (lgn, col).
3. Écrire une fonction `voisins_laby_inf` qui n'affiche rien et retourne une liste de couples d'entiers représentant les coordonnées des cellules voisines de (lgn, col).
4. Supposons, à présent, que le labyrinthe n'est pas infini et que :

$$\begin{aligned} 0 \leq \quad lgn \quad &< \text{nb\_lignes} \\ 0 \leq \quad col \quad &< \text{nb\_colonnes} \end{aligned}$$

5. Écrire une fonction `estDedans` qui, étant donnés deux entiers `lgn` et `col`, et les dimensions du labyrinthe (`nb_lignes` et `colonnes`), retourne `True` si la cellule `(lgn,col)` appartient au labyrinthe et `False` dans le cas contraire.
6. Quel est le nombre de voisins d'une cellule selon sa position ? (Inutile d'écrire une fonction)
7. Écrire une fonction `voisins_laby_fin` qui réalise la même tâche que la fonction `voisins_laby_inf` mais dans la situation d'un labyrinthe de dimensions finies `nb_lignes` et `nb_colonnes`. La fonction `voisins_laby_fin` prend en argument la position `lgn` et `col` de la cellule dont on veut connaître les voisins, ainsi que `nb_lignes` et `nb_colonnes`.

4 Le codage des caractères

4.1 Le codage par des entiers

En *python*, comme de nombreux langages, chaque caractère est associé à un entier qui représente le *code* de ce caractère. Il existe plusieurs types de code. L'étude de ces codes sort du cadre de ce TD. Python met à notre disposition :

— une fonction `ord` qui, étant donné un caractère, retourne son code (entier) ;

— une fonction `chr` qui, étant donné un entier, retourne le caractère correspondant au code. Par exemple le code de `'a'` est 97 et celui de `'z'` et de 122. Le code de toutes les lettres minuscules sont compris entre 97 et 122.

```
>>> ord('a')
97
>>> chr(97)
'a'
>>> chr(98)
'b'
>>> chr(99)
'c'
>>> ord('z')
122
```

Écrire une fonction `estMinuscule` qui, à partir d'une chaîne de caractères comprenant un seul caractère retourne `True` si ce caractère est une lettre minuscule et `False` dans tous les autres cas.

4.2 Pour aller plus loin : Le chiffrement de *César*

4.2.1 Principe

L'objectif est de chiffrer un texte pour le rendre illisible à toute personne qui ne possède pas la clef de lecture de ce code. Le chiffrement dit de *César* s'appelle aussi le chiffrement par *décalage*. Il consiste à parcourir les lettres d'un texte et de remplacer chaque lettre par une lettre qui se trouve à `n` crans plus loin. Si le décalage dépasse la lettre `'z'`, alors on repart au début de l'alphabet. Par exemple dans la figure ci-dessous, le décalage est de 3. Il s'ensuit que :

— Tous les `'a'` sont donc chiffrés par des `'d'` ;

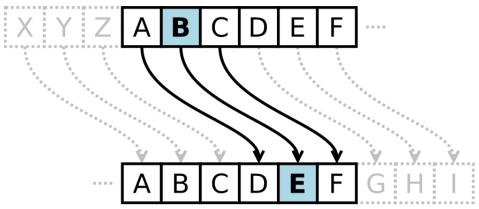
— Tous les `'b'` sont donc chiffrés par des `'e'` ;

— ...

— Tous les `'x'` sont donc chiffrés par des `'a'` ;

— Tous les `'y'` sont donc chiffrés par des `'b'` ;

— Tous les `'z'` sont donc chiffrés par des `'e'` ;



Chiffrement de *César*

Mais on peut choisir d'autres décalages, positifs ou négatifs.

Pour simplifier, on ne chiffre que les lettres minuscules. Ainsi avec un décalage de 4, un texte comme :

```
Bonjour, comment tu vas ?
Moi, je vais tres bien !
```

une fois chiffré, devient :

```
Bsrnsyv, gsqqirx xy zew ?
Msm, ni zemw xviw fmir !
```

Pour le déchiffrement, il suffit de faire le décalage dans le sens inverse.

4.2.2 Chiffrement

1. Écrire une fonction `decalageCar` qui, à partir d'une chaîne de caractères comprenant un seul caractère et d'un entier `decal`, positif ou négatif, représentant la valeur du décalage, retourne une autre chaîne de caractères représentant la lettre décalée. Vous pourrez utiliser les fonctions `ord` et `chr`. Par exemple :
   — `decalageCar('a',4)` retourne `'e'` ;
   — `decalageCar('p',4)` retourne `'t'` ;
   — `decalageCar('z',4)` retourne `'d'`.
   — `decalageCar('A',4)` retourne `'A'` (car `'A'` n'est pas une lettre minuscule)