Université de Strasbourg
Première année de licence
Semestre 1

Algorithmique et Programmation 1

TP5 : Fonctions

1 Trouver l'entrée et la sortie

Dans le TD5, vous avez écrit une fonction `indice` qui, à partir d'une liste d'entiers `lst` et d'un entier `n` retourne un entier qui vaut -1 si aucun élément de `lst` ne vaut `n`. Dans le cas contraire, l'entier retourné est l'indice du premier élément de `lst` qui vaut `n`.

Importer le module `Labyrinthe` et créer un labyrinthe (pas nécessairement très grand). Soit `laby` la liste de listes d'entiers représentant ce labyrinthe. On cherche à trouver la cellule représentant l'entrée et la sortie

1. Écrire et tester cette fonction ;
2. Écrire une fonction `coord` qui, à partir d'une liste de listes d'entiers, `laby`, et d'un entier `n` ne retourne rien si aucun des entiers de `laby` ne vaut `n`. Dans le cas contraire, la valeur retournée est un couple d'entiers représentant les coordonnées de l'élément valant `n`. Par exemple Par exemple si `laby` vaut :

   ```
   [ [1,2,3],
     [4,5,6] ]
   ```

   alors `coord(laby,3)` vaut `(0,2)` car l'élément qui vaut 3 est sur la ligne numéro 0 et la colonne numéro 3.
3. Importer le module `Labyrinthe` et créer un labyrinthe (pas nécessairement très grand). Le labyrinthe est une liste de listes d'entiers valant 0 (murs), 1 (couloirs), 2 (entrée) et 3 (sortie).
   - Écrire une fonction `entree` qui, à partir d'un labyrinthe (liste de listes d'entiers) retourne un couple d'entiers représentant les coordonnées de la cellule d'entrée ;
   - Écrire une fonction `sortie` qui, à partir d'un labyrinthe (liste de listes d'entiers) retourne un couple d'entiers représentant les coordonnées de la cellule de sortie.

     Dans les deux cas, vérifier, en affichant le labyrinthe, que votre résultat est bien conforme à l'entrée et sortie réelles du labyrinthe. Par exemple dans le cas du labyrinthe ci-dessous,

     ```
     [[0, 0, 0, 0, 0, 0, 0],
      [0, 2, 1, 1, 0, 1, 0],
      [0, 1, 0, 1, 0, 1, 0],
      [0, 1, 0, 1, 1, 3, 0],
      [0, 0, 0, 0, 0, 0, 0]]
     ```

     `entree(laby)` devrait retourner `(1,1)` pour l'entrée et `(3,5)` pour la sortie.

2 Les voisins d'une cellule

1. Dans le TP4, section 3, vous avez écrit des programmes permettant de déterminer le nombre de lignes et le nombre de colonnes d'une liste de listes. En déduire à présent deux fonctions : `nbLignes` et `nbColonnes` qui, à partir d'une liste de listes `laby` (on suppose que toutes les listes de `laby` ont la même longueur) retournent respectivement le nombre de lignes (le nombre de listes) et le nombre de colonnes (le nombre d'élément dans chaque liste).

2. Créer un labyrinthe et tester ces fonctions sur ce labyrinthe. Dans le cas du labyrinthe ci-dessus, `nbLignes(laby)` doit retourner `5` et `nbColonnes(laby)` doit retourner 7.
3. Dans le TD5, vous avez écrit une fonction appelée `voisins_laby_fin` qui, à partir de deux entiers `lgn` et `col` représentant les coordonnées d'une cellule, et des dimensions du labyrinthe, `nb_lignes` et `nb_colonnes`, retourne une liste de couples d'entiers représentant les coordonnées des cellules voisines de (lgn, col). Écrire cette fonction et la tester sur un vrai labyrinthe, en prenant une cellule du coin, du bord ou du centre du labyrinthe.
4. Dans le contexte d'un labyrinthe, nous serons souvent intéressés non pas par toutes les cellules voisines mais seulement les cellules accessibles (des cellules qui ne sont pas des murs). Écrire une fonction `voisins_laby_acc` qui, à partir de deux entiers `lgn` et `col`, représentant les coordonnées d'une cellule, et du labyrinthe lui-même, retourne une liste de couples d'entiers représentant les coordonnées des cellules voisines de (lgn, col) et qui ne sont pas des murs. Écrire cette fonction et la tester sur un vrai labyrinthe. Par exemple, dans le cas du labyrinthe ci-dessus, `voisins_laby_acc(1,5,5,7,laby)` doit donner un seul couple `(2,5)` car le point `(1,5)` n'a qu'une seule voisine accessible.
5. Modifier l'entête de la fonction pour que les arguments ne soient pas deux entiers et le labyrinthe, mais un couple d'entiers (une cellule) et le labyrinthe.

3 Le format `pgm`

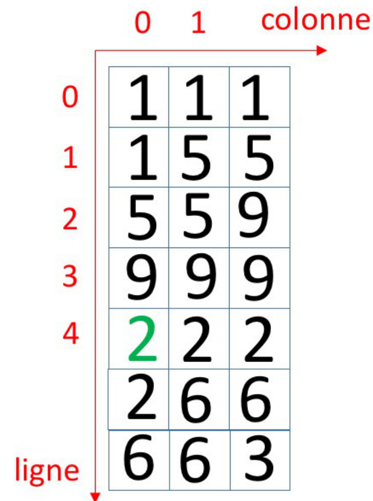
Pour des raisons décrites dans la section 5, nous ne souhaitons plus écrire la valeur des pixels directement dans le fichier, mais de les écrire dans une liste puis d'écrire la liste dans le fichier.

3.1 Dessiner un pixel dans une liste

Considérons un fichier image minuscule (3x7 pixels). Le fait de travailler avec un petit fichier permet de le voir en entier sous forme texte.

```
P2
3 7
255
1 1 1 1 5 5 5 5 9 9 9 9 2 2 2 2 6 6 6 6 3
```

Du au fait que le nombre de colonnes est de 3, il faut lire les entiers du fichier par groupes de 3. Ainsi la première ligne de pixels est constituée par les trois premiers entiers. La deuxième ligne est constituée par les trois entiers suivants et ainsi de suite. Le pixel qui se trouve à la colonne 0 et à la ligne 4 (dessiné en vert) correspond à l'entier d'indice 12. La question est : comment trouver l'indice (ici 12) en fonction des coordonnées du pixel (ici (4,0)) et réciproquement ?



La position des pixels correspondant au fichier ci-contre.

1. Écrire une fonction `indicePixel` qui, à partir d'un numéro de ligne `lgn`, d'un numéro de colonne `col` et du nombre total de colonnes `nb_colonnes` retourne l'indice du pixel `(lgn,col)` dans la liste des pixels. Par exemple, dans notre cas, `indicePixel(4,0,3)` devrait retourner 12.
2. Écrire une fonction `valeurPixel` qui, à partir de la liste de pixels `pixels`, d'un numéro de ligne `lgn`, d'un numéro de colonne `col` et du nombre total de colonnes `nb_colonnes`, retourne la valeur (l'intensité) du pixel `(lgn,col)` dans la liste. Par exemple, dans notre exemple, `valeurPixel(pixels,4,0,3)` devrait retourner 2.

3. Enfin, écrire une fonction `dessinePixel` qui, à partir de la liste des pixels `pixels`, d'un numéro de ligne `lgn`, d'un numéro de colonne `col`, du nombre total de colonnes `nb_colonnes`, et d'une valeur `val`, ne retourne aucune valeur mais change la valeur du pixel `(lgn,col)` pour le mettre à `val`. Par exemple `dessinePixel(pixels,4,0,3,0)` ferait du pixel `(4,0)` un pixel noir.

3.2 L'écriture : choix des arguments

Une fois la liste `pixels` d'entiers prête, il faut l'écrire dans un fichier. Comme nous l'avons dans plusieurs TPs, le programme suivant permet de créer un fichier image (appelée `toto.pgm` ici) avec 768 x 576 pixels, l'intensité des pixels étant donnée par la liste `pixels`.

```
f = open("toto.pgm","w")
f.write("P2\n")
f.write("768 576\n")
f.write("255\n")
for p in pixels:
    f.write(str(p) + " ")
f.close()
```

Comme c'est une opération que vous serez amenés à faire souvent, nous allons en faire une fonction `PGM_save`. Quels devraient être les arguments d'une telle fonction pour que n'ayez pas besoin de modifier son contenu selon les différents contextes où vous allez l'utiliser ? Écrire la fonction.

4 Pour aller plus loin : Le codage de *César*

Nous avons un fichier texte composé de mots et nous souhaitons chiffrer ce texte pour qu'il ne soit lisible que par la personne qui connaît la clef de lecture. Le principe est de décaler toutes les lettres minuscules d'un nombre constant de crans. Pour cela, nous avons besoin de :

— Savoir lire un fichier texte et mettre son contenu dans une chaîne de caractères ;

— Parcourir les caractères de la chaîne et appliquer un traitement à chacun d'eux ;

— Ce traitement consiste à reconnaître les lettres minuscules et à les décaler de n crans.

Dans le TD5, vous avez appris à faire les deux dernières opérations : reconnaître les minuscules et les décaler. Á présent, nous allons faire le chemin inverse. Autrement dit, nous allons en déduire une fonction qui peut décaler les lettres minuscules de tout un texte. Puis nous allons faire une fonction qui ouvre un fichier, lit le contenu et le traite et enregistre les résultats dans un autre fichier.

1. Écrire les fonctions `estMinuscule` et `decalageCar`, vues en TD et les tester.
2. Écrire une fonction `decalageStr` qui, étant donnée une chaîne de caractères `s` et un entier `decal` crée une nouvelle chaîne de caractères où les caractères non-minuscules sont les mêmes que dans `s` et où les caractères minuscules ont été décalées de `decal`. Par exemple :

   ```
   decalageStr("Bonjour !",4)
   ```

   doit retourner la chaîne `'Bsrnsyv !'`
3. Le programme suivant :

   ```
   f = open("entree.txt","r")
   contenu = f.read()
   f.close()
   ```

lit le contenu du fichier `entree.txt` et le met dans une chaîne de caractères (ici la variable `contenu`). En déduire une fonction `decalageFichier` qui, à partir d'un nom de fichier texte `intext`, d'un autre nom de fichier texte `outtext`, et d'une valeur de décalage, lit le fichier `intext`, décale les minuscules et enregistre le résultat dans `outtext`.

4. Tester cette fonction pour chiffrer et déchiffrer des fichiers textes.

5 Pour aller plus loin : écrire dans un fichier dans un ordre quelconque

Cette section ne contient aucune tâche à réaliser. Son objectif est de motiver les actions de la section 3.

L'écriture dans un fichier se fait de façon séquentielle

Dans les TPs précédents, vous avez déjà écrit dans des fichiers et, en particulier, dans des fichiers images. Vous l'aurez donc sans doute remarqué : l'écriture dans un fichier est *séquentielle*. Autrement dit, il faut commencer par le début du fichier et terminer par la fin. Après l'écriture d'un caractère en fin de fichier, on ne peut pas revenir en arrière et changer le premier caractère sans toucher au reste. Dans les fichiers images, cela impliquerait qu'on dessine toujours les pixels en haut à gauche pour terminer avec les pixels en bas à droite. Cette méthode convient lorsqu'en parcourant les pixels, on peut décider la couleur définitive de chacun d'eux.

Besoin de pouvoir écrire dans un ordre quelconque

Pour dessiner un labyrinthe, on parcourt les pixels. Pour chaque pixel, on calcule la cellule correspondante, dont la valeur (nulle ou non-nulle) détermine la couleur du pixel. Donc l'écriture séquentielle n'est pas gênante. Par contre, pour dessiner la *solution* du labyrinthe, il n'en est pas de même. La solution du labyrinthe est une liste de cellules, celles qu'il faut parcourir pour aller de l'entrée à la sortie. Pour dessiner cette solution, on pourrait aussi parcourir tous les pixels et, pour chaque pixel p, parcourir toutes les cellules de la solution pour voir si p appartient à l'une d'elle. Mais ce faisant, l'écrasante majorité des pixels parcourus le sont inutilement. Une autre manière consiste à parcourir seulement les cellules de la solution et, pour chacune de ces cellules, de colorier les pixels correspondants. Mais les solutions de labyrinthe ne commencent pas toujours en haut à gauche de l'image. Cela implique donc qu'on puisse colorier les cellules dans un ordre a priori quelconque.

Passer par une structure de données intermédiaire

L'écriture dans un fichier est toujours séquentielle, mais l'écriture dans une liste peut s'effectuer dans n'importe quel ordre. Donc la méthode proposée à la section 3 consiste, en quelque sorte, de dessiner dans une liste d'entiers `(pixels)` puis, une fois l'écriture finie, d'écrire la liste `pixels` de façon séquentielle, dans le fichier.