Université de Strasbourg
Première année de licence
Semestre 1

Algorithmique et Programmation 1

TP2 : Les entrées/sorties et les boucles

1 Dessiner un sapin

Nous cherchons à de dessiner un sapin comme sur la figure 1. Malgré les apparences, ce n'est pas une tâche triviale. Une des compétences importantes de l'informaticien est celle de pouvoir décomposer une tâche complexe en plusieurs tâches simples et de les résoudre séparément quand c'est possible. Pour l'heure nous vous proposons une décomposition possible.

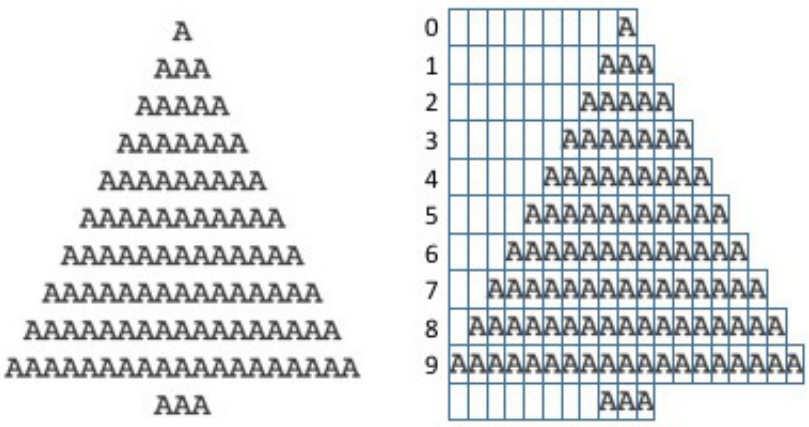

FIGURE 1 – On cherche à représenter un sapin (figure de gauche). Pour cela, on crée des chaînes de caractères composées d'espaces et de `'A'` (schématisé sur la figure de droite).

1. Écrire un programme `longueur.py` qui demande à l'utilisateur d'entrer un nombre et qui crée une chaîne de caractères composée d'autant de caractères `A` que le nombre spécifié par l'utilisateur. Par exemple si l'utilisateur entre 5, le programme devra créer la chaîne `"AAAAA"` et l'afficher.
2. Écrire un programme `longueurs2.py` qui demande à l'utilisateur d'entrer deux nombres. Le programme devra créer une chaîne de caractères composée d'espaces puis de `'A'`. Le nombre d'espace devra être égal au premier nombre entré par l'utilisateur et le nombre de `'A'` devra être égal au second nombre. Par exemple si l'utilisateur entre respectivement 2 et 7, alors la chaîne créée devra être `"  AAAAAAA"`
3. Écrire un programme `colonne.py` qui demande à l'utilisateur d'entrer un nombre. Le programme devra créer une chaîne de caractères composée de 5 caractères `'A'` et écrire cette chaîne autant de fois que le nombre entré par l'utilisateur. Par exemple si l'utilisateur entre 10, alors le programme devra afficher ce qui est représenté à gauche de la figure 2.
4. Écrire un programme `diagonale1.py` qui demande à l'utilisateur d'entrer un nombre de lignes `L`. Le programme devra afficher `L` lignes. La première ligne devra contenir 1 caractère `A`. La deuxième ligne devra contenir 2 caractères `A` et ainsi de suite jusqu'à ce que les `L` lignes aient été écrites (cf figure 2 au milieu).

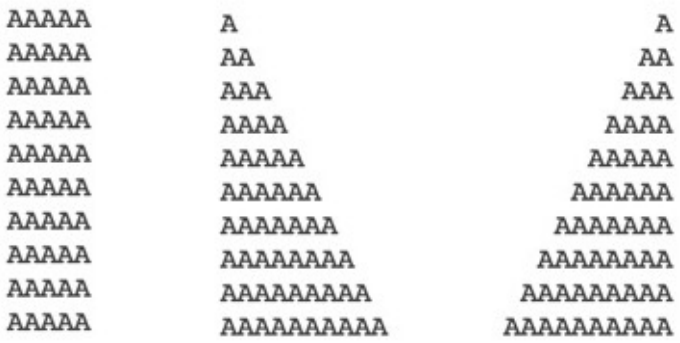

FIGURE 2 –

5. Écrire un programme `diagonale2.py` qui demande à l'utilisateur d'entrer un nombre de ligne `L`. Le programme devra afficher `L` lignes. Toutes les lignes contiennent `L` caractères. Sur la première ligne, il doit y avoir un caractère `A` à la dernière position. Sur la deuxième ligne, il doit y avoir deux caractères `A` à la fin et ainsi de suite jusqu'à ce que les `L` lignes aient été écrites (cf figure 2 à droite). La principale difficulté est de déterminer le nombre d'espaces et de caractères `A` à écrire sur chaque ligne.
6. Écrire un programme `sapin.py` qui demande à l'utilisateur d'entrer un nombre de ligne `L`. Le programme devra afficher `L` lignes. La première devra contenir un caractère `A`. La deuxième devra en contenir 3 et ainsi de suite jusqu'à ce que les `L` lignes aient été écrites. Pour finir le programme devra dessiner une ligne contenant trois caractères `A` et représentant le tronc du sapin (cf figure 1).

2 Écriture dans un fichier

Essayez le programme suivant :

```
fichier = open("message.txt","w")
fichier.write("Bonjour !\n Ca va ?")
fichier.close()
```

En regardant dans le répertoire où se trouve votre programme, vous découvrirez un nouveau fichier appelé `message.txt`. Je vous laisse ouvrir ce fichier et découvrir son contenu.

— La première ligne du programme constitue une *ouverture* du fichier `message.txt`. Le second argument `'w'` signifie que ce fichier a été ouvert en vue d'une écriture. (Il est aussi possible d'ouvrir un fichier pour le lire). Cette instruction produit une variable (ici `fichier`) qui nous permettra d'agir sur ce fichier.

— La deuxième instruction écrit le message `"Bonjour !\n Ca va ?"` dans ce fichier.

— La dernière instruction finalise l'écriture et, dit-on, *ferme* le fichier.

Essayez encore le programme suivant :

```
fichier = open("message.txt","w")
fichier.write("Bonjour !\n Ca va ?")
fichier.close()
# Arrive a ce point, le fichier message.txt contient "Bonjour ! Ca va ?"
fichier = open("message.txt","w")
fichier.write("Il fait beau chez vous ?")
fichier.close()
# Quel est le contenu de message.txt maintenant ?
```

et comparer son résultat avec :

```
fichier = open("message.txt","w")
fichier.write("Bonjour !\n Ca va ?")
fichier.write("Il fait beau chez vous ?")
fichier.close()
```

À présent, vous êtes capable de créer des fichiers et d'y écrire tous les textes que vous souhaitez.

3 Projet labyrinthe : écrire des fichiers image

3.1 Les formats *portable pixmaps*

Il existe de très nombreux formats d'images. Les uns rivalisent avec les autres pour représenter les images sur des fichiers les plus petits possibles avec le moins de perte d'information possible. Nous utiliserons ici le format `pgm` qui n'effectue aucune compression (et aucune perte d'information). Il fait partie d'une famille de formats (ppm, pgm, pbm) décrits plus en détail sur ce site `https://fr.wikipedia.org/wiki/Portable_pixmap`.

— le format `pbm` pour les images en noir et blanc ;

— le format `pgm` pour les images en niveaux de gris ;

— le format `ppm` pour les images en couleur.

L'avantage de ces formats est qu'il permet de représenter une image sous la forme d'un fichier texte. On pourrait donc le lire et le modifier avec un simple éditeur de texte.

3.2 Le format `pgm`

Un fichier au format `pgm` :

— commence toujours par le mot `P2` suivi d'un espace ou d'un retour à la ligne. Ces deux caractères sont appelés le *magic number* et permettent aux applications de lecture et d'édition d'image de reconnaître une image au format `pgm`

— doit ensuite comporter deux entiers représentant respectivement la largeur et la hauteur de l'image en pixels ;

— doit encore comporter un entier représentant l'intensité du blanc (généralement 255) ;

— tous les entiers qui suivent sont interprétés comme la valeur des pixels de l'image.

Par exemple le fichier suivant :

```
P2
10 5
255
127 127 127 127 127 127 127 127 127 127 0 0 0 0 0 0 0 0 0 0 255 255 255 255 255
255 255 255 255 255 200 180 160 140 120 100 80 60 40 20
```

représente une image minuscule de 10 pixels de large et 5 pixels de haut. Le blanc est représenté par 255. Ces trois premières lignes sont appelées *l'entête* du fichier image. Ensuite viennent une suite d'entiers représentant l'intensité de tous les pixels. La première valeur (ici 127) représente le pixel en haut à gauche de l'image. Ensuite viennent les pixels de la première ligne puis ceux de la deuxième ligne et ainsi de suite. L'image ci-dessus comporte une ligne de pixels (10 pixels) à 127, c'est à dire gris moyen. La deuxième ligne (10 pixels suivants) sont à 0 (donc noir). Les 10 pixels suivants sont à 255 (donc blanc) et la quatrième ligne représente un dégradé allant du presque blanc (200) à du presque noir (20). Il n'y a aucune valeur pour la dernière ligne. Elle est interprétée comme étant noire. Le retour à la ligne n'a aucune signification.

Pour vous convaincre, lancer votre éditeur de texte préféré. Ouvrir un nouveau document et copier-coller le contenu ci-dessus dans ce fichier. L'enregistrer avec un nom ayant une extension `.pgm` (supposons que vous l'ayez appelée `premier.pgm`). Ca y est ! Vous avez créé votre premier fichier image. Pour le voir vous pouvez :

— soit trouver le fichier dans un navigateur de fichier et double-cliquer dessus ;

— soit, avec un terminal, aller dans le même répertoire que le fichier image et entrer : `eog premier.pgm`

`eog` *(Eye Of Gnome)* est une application installée sur les machines de l'UFR et qui permet de voir la plupart des formats d'image. Attention, n'oubliez pas qu'il s'agit d'un fichier de 10 pixels sur 5. Donc il faudra beaucoup zoomer pour voir les détails de l'image.

Pour vous assurer que vous avez compris le format des fichiers `pgm`, ouvrir un éditeur de texte et créer les images suivantes de dimension 8 x 6 pixels.

1. un fichier `grisPetit.pgm` constitué seulement de pixels gris moyen (127) comme sur la figure 3.a ;
2. un fichier `NB_horizPetit.pgm` constitué de 3 lignes blanches et de 3 lignes noires comme sur la figure 3.b ;
3. un fichier `NB_verticPetit.pgm`. Sur chaque ligne, les 4 premiers pixels sont blancs et les 4 autres sont noirs comme sur la figure 3.c ;
4. un fichier `degradeHorizPetit.pgm`. Sur chaque ligne, le premier pixel est noir, les pixels suivants augmentent linéairement jusqu'au dernier pixel qui doit être blanc (255) comme sur la figure 3.d ;

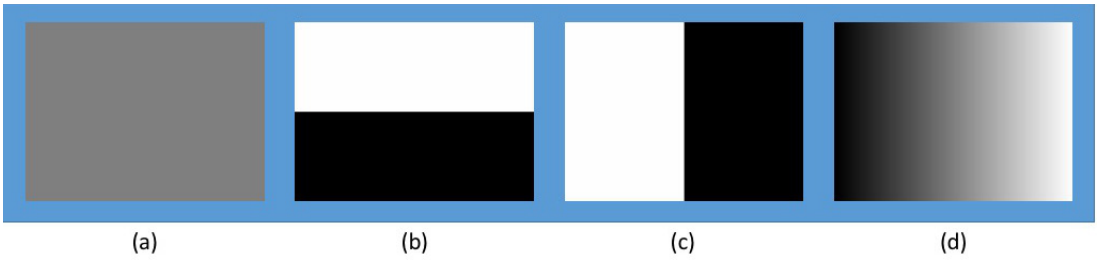

FIGURE 3 –

3.3 Écrire un fichier image avec *python*

Les exercices suivants consistent à créer des fichiers image de 768×576 pixels. On ne peut plus écrire des fichiers de cette taille à la main comme ci-dessus. C'est là où *python* nous sera d'une aide précieuse. Je vous demande de ne pas utiliser les conditionnels (`if, else, while`), mais de vous inspirer de ce que vous avez déjà fait pour le sapin. Pour chaque tâche, faire un programme python séparé.

1. Écrire un programme *python* qui crée un fichier image appelé `gris.pgm` de 768×576 pixels entièrement composée de pixels gris moyen (127) (cf figure 3.a) ;
2. Écrire un programme *python* qui crée un fichier image appelé `NB_horiz.pgm` de 768×576 pixels composée de pixels blancs (255) sur la moitié supérieure et des pixels noirs (0) sur la moitié inférieure (cf figure 3.b) ;
3. Écrire un programme *python* qui crée un fichier image appelé `NB_vertic.pgm` de 768×576 pixels composée de pixels blancs sur la moitié gauche et des pixels noirs sur la moitié droite (cf figure 3.c)
4. Écrire un programme *python* qui crée un fichier image appelé `degradeHoriz.pgm` de 256×192 pixels représentant un dégradé horizontal du noir vers le blanc (cf figure 3.d) ;