# TP noté : arbres généalogiques.

PTSI Lycée Eiffel

décembre 2014

Ce TP servira d'évaluation sur machine pour le premier semestre. Vous ferez le TP seul ou à deux selon le nombre de machines disponibles pour votre groupe. Il est évident que les élèves travaillant en binôme ont le droit de communiquer entre eux, mais de façon suffisamment discrète pour ne pas perturber le travail de leurs camarades. L'enseignant surveillant le TP n'interviendra que pour résoudre d'éventuels problèmes techniques, mais ne pourra pas aider les étudiants à répondre aux questions posées dans ce TP. Il est autorisé de sauter une ou plusieurs questions, en veillant dans ce cas à indiquer clairement les questions traitées.

- **Les notes de cours sont autorisées (mais PAS le recours à des programmes tout faits trouvés sur Internet).**
- **Les seules fonctions Python sur les listes que vous avez le droit d'utiliser sont *len* et la méthode *append*. En particulier, PAS de *sort* ou de *sum*.**
- **Les étudiants rendront à la fin du TP une copie papier (certaines questions nécessitent d'être traitées sur papier) ainsi qu'un fichier Python (extension .py) contenant leurs programmes que l'enseignant récuperera directement sur une clé USB. Il est bien entendu autorisé et même conseillé de commenter les programmes écrits sur ce fichier (les commentaires en Python s'introduisent avec un #).**

## I. Programmation de quelques algorithmes élémentaires sur les listes.

Cette section, indépendante des suivantes, propose d'écrire en Python, en guise d'échauffement, quelques programmes classiques faisant intervenir des listes. Les fonctions écrites dans cette section, si elles ont été correctement programmées, pourront être réutilisées dans les programmes demandés dans la suite du TP. Pour illustrer les questions et vous permettre de vérifier que vos programmes tournent correctement, on utilisera dans l'énoncé la liste exemple liste=[2,7,8,1,2,7,2,3,4,7,1].

**Question 1** : Écrire une fonction Python **appartient(l,a)** déterminant si un élément a apparait dans une liste l. La valeur retournée devra être un booléen (True ou False), mais si vous n'arrivez qu'à faire une version retournant un message, donnez-là quand même.

Pour cette première question, on vous donne un algorithme en pseudo-langage, que vous n'aurez qu'à retranscrire en Python. Rien ne vous interdit toutefois de créer votre fonction sans vous référer à cette aide.

```
définition de la fonction appartient(l,a) :
    pour i variant dans la liste l :
        si i est égal à a :
            retourner la valeur True
        sinon :
            on ne fait rien
    retourner la valeur False
```

La dernière ligne ne doit bien sûr être exécutée qu'une fois la boucle terminée.

**Exemple : appartient(liste,3)** doit retourner la valeur True ; **appartient(liste,6)** doit retourner False.

**Question 2** : Écrire une fonction Python **occurrences(l,a)** déterminant le nombre de fois où l'élément a apparait dans la liste l (la fonction renvoit 0 si l'élément n'est pas dans la liste).

**Exemple : occurrences(liste,2)** doit retourner la valeur 3.

**Question 3** : Écrire une fonction Python **plusgrands(l,m)** déterminant tous les éléments de l qui sont supérieurs ou égaux à m. La fonction renverra donc une liste, qui sera vide si tous les éléments de l sont strictement inférieurs à m.

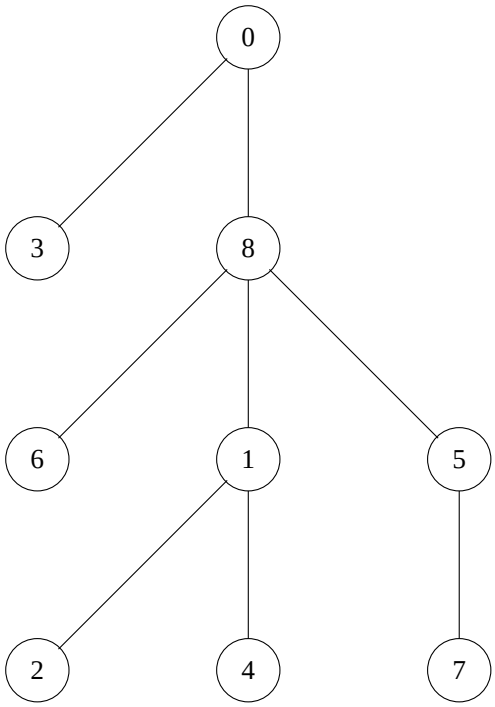
**Exemple : plusgrands(liste,5)** doit retourner la liste [7,8,7,7].

**Question 4** : Écrire une fonction Python **indices(l,a)** déterminant les indices de la liste où apparait la valeur a. La fonction renverra une liste, vide dans le cas où a n'apparait jamais dans la liste.

**Exemple : indices(liste,7)** doit retourner la liste [1,5,9].

## II. Présentation du problème : modélisation d'arbres généalogiques.

Le but de la suite du TP est de modéliser (et programmer quelques fonctionnalités sous Python) une structure élémentaire représentant un arbre généalogique. On commence par représenter l'arbre sous forme d'un graphe : en haut de l'arbre se situe l'ancêtre commun à tous les membres de l'arbre généalogique, et chaque membre de l'arbre est relié à chacun de ses enfants. Par souci de simplicité, les conjoints ne sont pas représentés dans l'abre. On suppose également qu'il s'agit d'un arbre « descendant » où on représente sur quelques générations la descendance issue d'un unique ancêtre commun. Ainsi, chaque membre de l'arbre est relié vers le haut à une unique personne (qu'on appelera simplement son **père**, sans se préoccuper du sexe des différents membres de l'arbre), à l'exception de l'ancêtre commun tout en haut de l'arbre qui n'a pas de père, et vers le bas à tous ses enfants (dont le nombre est un entier pouvant évidemment être nul, et pour lequel on ne fixera pas de borne supérieure). Prenons un exemple pour plus de clarté :




L'ancêtre commun sera toujours numéroté 0, le reste de la numérotation peut par contre être faite dans n'importe quel ordre (mais doit utiliser les entiers entre 0 et $n-1$, où $n$ est le nombre total de membres dans l'arbre). Ici, cet ancêtre a deux enfants (numérotés 3 et 8). Le membre 3 n'a pas d'enfant, mais 8 en a trois (numérotés 6, 1 et 5) etc. On a ici fait un arbre sur trois générations (on considère que l'ancêtre appartient à la génération 0, ses enfants à la génération 1, etc). La structure de l'arbre généalogique est entièrement déterminée par la donnée du père de chacun de ses membres (on décrète par convention que l'ancêtre 0 a pour père le numéro $-1$ qui ne correspond à personne dans l'arbre). Ainsi, on peut représenter l'arbre précédent par le tableau à double entrée suivant :

| Membre de l'arbre | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| Père du membre | -1 | 8 | 1 | ? | ? | ? | ? | ? | ? |

En Python, on représentera l'arbre sous forme d'une liste d'entiers contenant simplement la deuxième ligne du tableau précédent (autrement dit la liste des pères des membres de l'arbre, par ordre de numéro de membre). Ainsi, la liste Python représentant notre arbre-exemple sera la liste **arbre=[-1,8,1,?,?,?,?,?,?]**.

**Question 5** (sur papier) : Compléter le tableau précédent et donner la liste **arbre** complète.

**Question 6** (sur papier) : Tracer l'arbre généalogique correspondant à la liste arbre2=[-1,2,8,2,5,1,1,8,0]. Combien de générations sont présentes sur cet arbre ?

## III. Programmation de quelques fonctionnalités simples sur les arbres généalogiques.

Commençons par préciser un peu de vocabulaire sur nos arbres généalogiques : nous avons déjà défini les termes d'ancêtre (l'élément en haut de l'arbre), de père et d'enfant, et de génération. Plus

généralement, un élément a de l'arbre est un descendant d'un autre élément b si on peut passer de a à b par un chemin entièrement descendant dans l'arbre (autrement dit, b est un enfant de a, ou un enfant d'un des enfants de a, etc). On convient qu'un élément est toujours son propre descendant.

**Question 7** : Écrire une fonction Python **pere(l,a)** déterminant le père de l'élément a dans la liste l (qui sera supposée représenter un arbre généalogique).

**Exemple : pere(arbre,5)** doit retourner la valeur 8.

**Question 8** : Écrire une fonction Python **enfants(l,a)** déterminant la liste des enfants de l'élément a dans la liste l.

**Exemple : enfants(arbre,8)** doit retourner la liste [1,5,6].

**Question 9** : Écrire une fonction Python **generation(l,a)** déterminant à quelle génération appartient l'élément a dans la liste l

**Exemple : generation(arbre,6)** doit retourner la valeur 2.

**Question 10** : Écrire une fonction Python **nbredescendants(l,a)** déterminant le nombre de descendants de l'élément a dans la liste l (on pourra commencer par écrire une fonction descendant(l,i,j) déterminant si j est un descendant de i dans la liste l).

**Question 11** : Écrire une fonction Python **nbregenerations(l)** déterminant le nombre de générations présentes dans l'arbre représenté par la liste l.

**Exemple : nbregenerations(arbre)** doit retourner la valeur 3.

## IV. Des fonctionnalités plus avancées.

Continuons avec le vocabulaire : si a et b sont deux membres d'un arbre généalogique, leur plus proche ancêtre est l'élément de l'arbre appartenant à la génération la plus grande ayant pour descendants à la fois a et b (un tel plus proche ancêtre existe toujours, et il est même unique). On définit la distance de a à b comme étant la longueur du plus court chemin menant de a à b dans l'abre généalogique. Ce chemin est toujours celui partant de a et remontant jusqu'au plus proche ancêtre commun, puis redescendant jusqu'à b (ainsi, deux éléments qui sont les fils d'une même personne seront à distance 2).

**Question 12** : Écrire une fonction Python **plusprocheancetre(l,a,b)** déterminant le plus proche ancêtre commun de a et b dans la liste l (on pourra commencer par traiter le cas où a et b sont de la même génération).

**Exemple : plusprocheancetre(arbre,6,4)** doit retourner la valeur 8.

**Question 13** : Écrire une fonction Python **distance(l,a,b)** déterminant la distance entre a et b dans la liste l.

**Exemple : distance(arbre,3,7)** doit retourner la valeur 4.

**Question bonus** : Écrire une fonction Python **genealogique(l)** qui détermine si une liste d'entiers en Python (quelconque) est une liste représentant un arbre généalogique ou non.

**Exemple : genealogique([-1,3,4,4,1,0])** doit retourner la valeur False.