TP PYTHON - Quelques exercices d'algorithmique

Exercice 1:

1. Ecrire une fonction qui étant donné un entier n renvoie la liste de ses diviseurs stricts, ie tous ses diviseurs excepté lui-même. A l'aide de cette fonction, écrire une fonction *premier* qui étant donné un entier n renvoie True s'il est premier et False sinon.
2. Un nombre est dit :
   •**parfait** si la somme de ses diviseurs stricts est égale à ce nombre.
   •**abondant** si la somme de ses diviseurs stricts est strictement supérieure à ce nombre.
   •**déficient** si la somme de ses diviseurs stricts est strictement inférieure à ce nombre.
   Par exemple, 6 est parfait, 12 est abondant, 5 est déficient.
   Écrivez une fonction qui étant donné un entier n renvoie si celui-ci est parfait, abondant ou déficient.
3. Déterminer le plus petit nombre abondant impair.
4. Quelle proportion des nombres entre 1 et 1000 sont abondants ? déficients ?

Dans les exercices suivants on pourra utiliser les fonctions suivantes :

```
1 def liste_entier(n) :
2     """ Cette fonction transforme un entier en une liste de ses chiffres utilisées pour l'écriture en base 10 de cet entier. Ainsi 145 devient la liste [1,4,5] """
3     x=str(n) # on transforme l'entier n en string
4     p=len(x) # nombre de chiffres utilisés pour n
5     L=[] # initialisation d'une liste vide
6     for k in range(p) :
7         L.append(int(x[k])) # A l'étape k, on ajoute à la liste le kième élément de x que l'on type à nouveau en entier
8     return(L)
```

```
1 def entier_liste(L) :
2     """ Cette fonction transforme une liste d'entiers en un entier dont l'écriture en base 10 correspond aux éléments de la liste. Ainsi la liste [4,2,1] devient 421 """
3     p=len(L) # taille de la liste
4     n=0 # initialisation
5     for k in range(p) :
6         n+=10**k*L[p-1-k] # attention le chiffre des unités est le dernier de la liste L, ie celui d'indice p-1
7     return(n)
```

Exercice 2: Le problème est d'afficher tous les nombres de 3 chiffres qui sont égaux à la somme du cube de leurs chiffres. Ce sont les nombres d'Armstrong.
Exemple : $153 = 1^3 + 5^3 + 3^3$
L'idée est de faire parcourir un entier parmi ceux à 3 chiffres, de calculer la somme du cube de ses chiffres et de stocker les entiers solutions.

Exercice 3: Les nombres 46 et 96 possèdent une propriété intéressante : leur produit ne change pas lorque l'on permute leurs chiffres. En effet : $46 * 96 = 4416 = 64 * 69$
Ecrire une fonction qui renvoie tous les couples de nombres de deux chiffres possédant la même propriété.

Exercice 4:

1. Ecrire une fonction *produit*(n) qui étant donné un entier n renvoie le produit des chiffres composant n. Par exemple si n=175, alors *produit*(n) doit retourner $1 * 7 * 5 = 35$
2. On appelle persistance de n l'entier égal au nombre de fois qu'il faut itérer la fonction *produit* pour obtenir un nombre inférieur ou égal à 9 (après, la suite est stationnaire). Par exemple la persistance de 175 est 3 car *produit*(175)=35, puis *produit*(35)=15 et *produit*(15)=5.
   Ecrire alors la fonction *persistance*(n) qui étant donné un entier n renvoie la persistance de n.
3. Ecrire un script affichant la liste des persistances de tous les entiers entre 11 et 100 puis parmi eux l'entier avec la persistance maximale.

Exercice 5: On remarque que $19^2 + 89^2 = 8282$.
Ecrire une fonction déterminant le(s) nombre(s) bègues (à 4 chiffres de la forme *abab*) égaux à la somme de deux carrés.

Exercice 6: On décide de jouer avec les nombres de la manière suivante : on se donne un nombre, on ajoute les carrés des chiffres qui le composent, on recommence avec le nombre obtenu, et on continue... Prenons par exemple le nombre 153, il devient successivement $1^2 + 5^2 + 3^2 = 35$ puis $3^2 + 5^2 = 34$ puis ...
Construire la fonction *carre*(n) permettant de construire le nombre obtenu par ce processus s'il s'arrête (au sens de produire deux fois de suite le même nombre) ou bien celui obtenu à la 10 000ème itération.

Exercice 7:
Ecrire une fonction déterminant le(s) nombre(s) à 4 chiffres de la forme *abcd*, vérfifiant :

$$\begin{array}{cccc} & b & d & a & c \\ + & & & & \\ \hline & d & c & b & a \\ \hline & a & b & c & d \end{array}$$

Exercice 8: Un carré magique est une matrice de n lignes et n colonnes, dont les coefficients sont les entiers appartenant à $[\![1, n^2]\!]$ (chaque entier apparaissant une et une seule fois) et telle que les sommes des coefficients de chaque ligne, de chaque colonne et des deux diagonales du carré sont toutes les mêmes. Par exemple :

4	3	8
9	5	1
2	7	6

est un carré magique : en effet, les sommes des coefficients de chaque ligne, de chaque colonne et des deux diagonales sont toutes égales à 15, et chaque entier de $[\![1, 9]\!]$ apparaît une et une seule fois.

1. Écrire une fonction, étant donnés une matrice M et le numéro i d'une ligne de M, renvoie la somme des éléments de la ligne i de M.
2. Écrire une fonction qui permet de savoir, pour une matrice carrée M donnée en argument d'entrée, si les sommes de chaque ligne, de chaque colonne et des deux diagonales sont toutes les mêmes.
3. Écrire une fonction qui permet de savoir, pour une matrice carrée M donnée en argument d'entrée, si chaque élément de $[\![1, n^2]\!]$ apparaît une et une seule fois.
4. Écrire enfin une fonction qui permet de savoir, pour une matrice carrée M donnée en argument d'entrée, si c'est un carré magique ou pas.

Exercice 9:
Le jeu de Nim : le jeu se joue à 2, c'est chacun son tour de jouer. Le tapis comporte une seule rangée de 21 allumettes. Lorsque c'est son tour de jouer, un joueur doit enlever une, deux ou trois allumettes. Celui qui retire la dernière allumette a perdu la partie.

1. Ecrire une fonction qui permet à deux utilisateurs de jouer l'un contre l'autre. A chaque étape, on demandera au joueur dont c'est le tour le nombre d'allumettes qu'il souhaite retirer, puis le nombre d'allumettes restant sur la table après son coup. On affichera à la fin qui est le vainqueur.
2. Ecrire une seconde fonction qui permet cette fois-ci de jouer contre l'ordinateur, celui-ci jouant de manière aléatoire à chaque coup.
3. Ecrire enfin une fonction qui fait gagner l'ordinateur à tous les coups dès lors que l'utilisateur commence !