Cahier de vacances d'informatique

1 Joli dessin

On donne la liste `mystere` qui contient un dessin mystère à révéler :

```
## le symbole \ permet
## d'écrire sur plusieurs
## lignes
mystere = [[2,7],[5,4],[6,0],[4,-3],[0,-4],\
           [-4,-3],[-6,0],[-5,4],[-2,7],[2,7],\
           [-1,4],[-1,2],[-3,2],[-3,4],[-1,4],\
           [1,4],[3,4],[3,2],[1,2],[1,4],\
           [0.5,1],[1,-1],[-1,-1],[-0.5,1],[0.5,1],[-2,-2],[0,-3],[2,-2]]
```

On rappelle que la commande `import matplotlib.pyplot as plt` (à ne taper qu'une seule fois avant utilisation) permet l'utilisation du module `pyplot` permettant les représentations graphiques sur Python. Pour tracer le segment $[AB]$ où $A=(x_A,y_A)$ et $B=(x_B,y_B)$, on écrira donc :

```
plt.plot([xA,xB],[yA,yB],'r') ## 'r' permet un tracé en rouge ('b' pour du
   bleu, 'g' pour du vert...)
```

On pourra aussi utiliser la commande `plt.axis('equal')` pour avoir un repère ortho-**normé**.

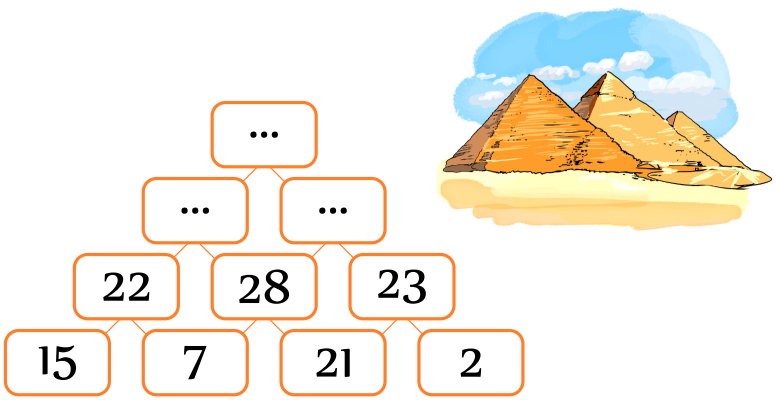
Faire le tracé du dessin mystère en sachant que :

- les dix premiers éléments de la liste `mystere` sont des points à relier entre eux ;
- les dix éléments suivants sont également des points à relier entre eux ;
- parmi les éléments restants :
  - ceux dont l'ordonnée est égale à -1 ou 1 sont à relier entre eux ;
  - ceux dont l'abscisse est paire sont à relier entre eux ;

Choisir une couleur pour chaque ensemble de point. Attention à bien conserver les points dans l'ordre donné dans la liste.

2 Pyramide d'addition

On veut écrire un code Pyhton permettant de calculer tous les termes d'une "Pyramide d'addition". Pour remplir une case de la pyramide, il faut additionner les cases directement sous celle-ci. La plupart du temps, seules les cases de la base sont remplies.



Le code proposé à pour objectif de créer une liste de listes représentant la pyramide. Chaque sous-liste contient un étage de la pyramide. Au départ, seul l'étage initial est saisi.

```
Pyramide = [[15,7,21,2]]

def calculPyramide(Pyramide):
    while len(Pyramide[-1])>1:
        Lsuivant = []
        for i in range(0,len(Pyramide[-1])-1):
            Lsuivant.append(... à compléter ...)
        ... à compléter ...
    return Pyramide

print(calculPyramide(Pyramide))
```

Q°1 : Compléter le code ci-dessus.

Q°2 : Écrire la procédure permettant de calculer le terme le plus en haut de la pyramide si la base est constituée de tous les entiers allant de 0 à 1000 (inclus). Tester votre code.

Q°3 : Quelle est la complexité de cet algorithme (on ne comptera que les additions effectuées) ?

Q°4 : En supposant que vous êtes capable de faire chaque addition en 10 secondes (*Je suis optimiste !*), de combien de temps auriez-vous besoin pour calculer la pyramide de la question 2.

3 Vérification d'un Sudoku

		1		3	4			6
	4				2	5	8	
	5		1	8			4	
1	3	2		4				
				1	6			
7	6	4	8	5			2	
			3	9	1		5	8
9	1		7		8	4		
6			4	2				7

Q°1 : Résoudre, à la main, le Sudoku ci-dessous (chercher les règles sur internet si besoin).

On va maintenant écrire un code Python permettant de vérifier que la grille remplie à la question précédente est correcte.

Q°2 : Représenter votre grille sous forme d'une liste de listes. Chaque sous-liste doit représenter une ligne de la grille.

Q°3 : Écrire une fonction `bonneliste(L)` qui renvoie `True` si la liste est une "bonne liste" et `False` sinon. On dira que `L` est une "bonne liste" si `L` contient chacun des entiers allant de 1 à 9 (inclus). On pourra définir une liste `compteur`, de longueur 9 et initialement remplie de 0, qui devient au fur et à mesure une liste remplie de 1 si la liste est une "bonne

liste".

Q°4 : Écrire une fonction `testligne(i,S)` qui teste que la ligne `i` de la grille `S` est une "bonne liste" (utiliser la fonction `bonneliste`).

Q°5 : Écrire une fonction `testcolonne(j,S)` qui teste que la colonne `j` de la grille `S` est une "bonne liste".

Q°6 : Écrire une fonction `testcarre(i,j,S)` qui teste que le carré centré sur la case ligne `i` et colonne `j` de la grille `S` est une "bonne liste".

Q°7 : Écrire une fonction `test(S)` qui teste votre grille entièrement.