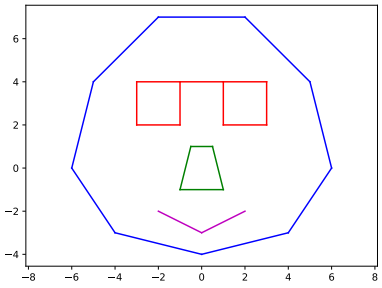
Correction du cahier de vacances d'informatique

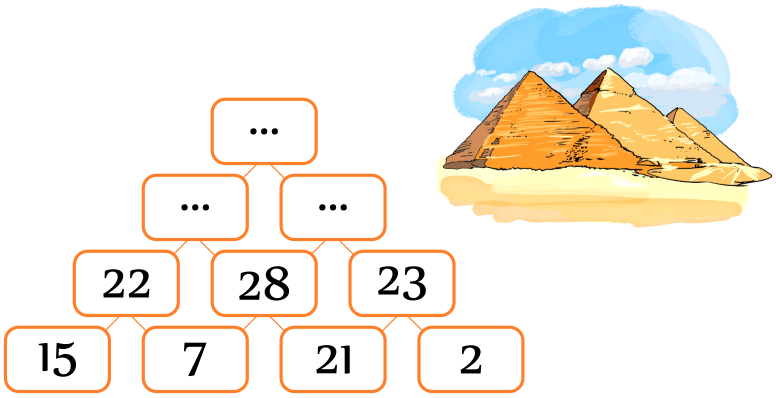
1 Joli dessin

```
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]]

import matplotlib.pyplot as plt

plt.axis('equal')

L = mystere

for i in range(0,9):
    plt.plot([L[i][0],L[i+1][0]],[L[i][1],L[i+1][1]],'b')

for i in range(10,19):
    plt.plot([L[i][0],L[i+1][0]],[L[i][1],L[i+1][1]],'r')


Ln = []
Lb = []

for i in range(20,len(L)):
    if L[i][1]==1 or L[i][1]==-1:
        Ln.append(L[i])
    if L[i][0]%2==0:
        Lb.append(L[i])

for i in range(0,len(Ln)-1):
    plt.plot([Ln[i][0],Ln[i+1][0]],[Ln[i][1],Ln[i+1][1]],'g')

for i in range(0,len(Lb)-1):
    plt.plot([Lb[i][0],Lb[i+1][0]],[Lb[i][1],Lb[i+1][1]],'m')
```



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.

```
bigPyramide = [i for i in range(0,1001)]
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(Pyramide[-1][i] + Pyramide[-1][i+1])
        Pyramide.append(Lsuivant)
    return Pyramide

print(calculPyramide(Pyramide))
```

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.

```
bigPyramide = [[i for i in range(0,1001)]]
```

```
print(calculPyramide(bigPyramide)[-1])
```

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

On appelle n, le nombre de nombres à la base de la pyramide. Pour calculer l'étage suivant, il faut $n-1$ additions. Pour remonter à nouveau d'un étage, il faut $n-2$ additions, puis $n-3$, etc. Au total, la complexité $c(n)$ est donc :

$$c(n) = \sum_{i=1}^{n} n-1 = \frac{n.(n-1)}{2}$$

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.

Le temps nécessaire est donc : $t_{1000} = 10.c(1000) \approx 58$ jours contre 0.2 secondes sur mon ordinateur (on peut mesurer une durée avec le module `clock` de `Python`).

3 Vérification d'un Sudoku

Q°1 : Solution du Sudoku

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

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.

```
S = [[8,7,1,5,3,4,2,9,6],[3,4,9,6,7,2,5,8,1],[2,5,6,1,8,9,7,4,3],\
     [1,3,2,9,4,7,8,6,5],[5,9,8,2,1,6,3,7,4],[7,6,4,8,5,3,1,2,9],\
     [4,2,7,3,9,1,6,5,8],[9,1,5,7,6,8,4,3,2],[6,8,3,4,2,5,9,1,7]]
```

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".

```
L = [1,2,3,4,5,6,7,8,9]
def bonneliste(L):
    compt = 9*[0]
```

```
    for i in  range(0,9):
        compt[L[i]-1] += 1
    for j in compt:
        if j!=1:
            return False
    return True
```

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`).

```
def testligne(i,S):
    L = S[i]
    return bonneliste(L)
```

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

```
def testcolonne(j,S):
    L = [S[i][j] for i in range(0,9)]
    return bonneliste(L)
```

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".

```
def testcarre(i,j,S):
    L2 =   S[i-1][j-1:j+2]+S[i][j-1:j+2]+S[i+1][j-1:j+2]
    return bonneliste(L2)
```

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

```
def test(S):
    for i in range(0,9):
        if not testligne(i,S):
            return False
        if not testcolonne(i,S):
            return False
    for i in range(1,8,3):
        for j in range(1,8,3):
            if not testcarre(i,j,S):
                return False
    return True
```