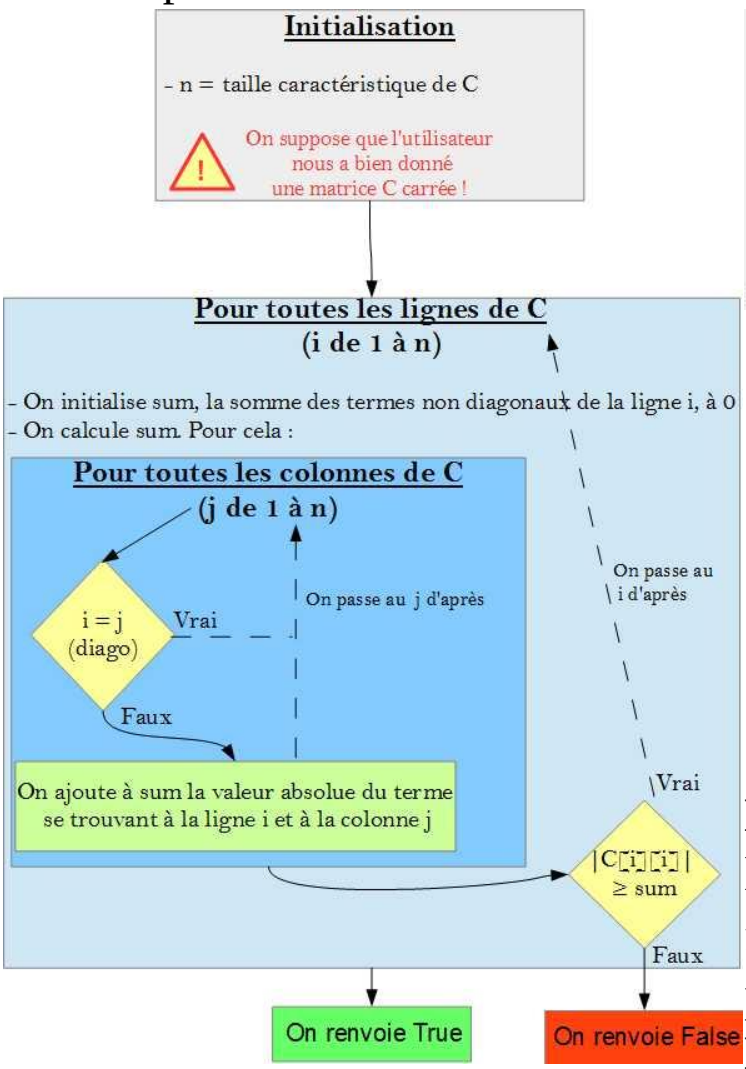
Exercice *M1/ ★ L'exercice consiste à créer une fonction **diagdom** qui a pour paramètre *une matrice A carrée* et qui renvoie True si la matrice est à diagonale dominante, ou False dans le cas contraire. (ici False et True sont des booléens, pas une chaîne de caractères)

Correction

- une solution fonctionnant sous Python 3.4 :

Code copiable : [Afficher]




```
1 def diagdom(C):
2     n = C.shape[0]
3     for i in range(n):
4         sum = 0
5         for j in range(n):
6             if j != i:
7                 sum += abs(C[i][j])
8         if abs(C[i][i]) < sum:
9             return(False)
10    return(True)
```

L1. On commence la définition de la fonction **diagdom** de paramètre C.

L2. On récupère le nombre de lignes (ou de colonnes) de C à l'aide de la méthode shape qui renvoie le tuple suivant : (nombre de lignes, nombre de colonnes) dont on ne récupère que le premier terme.

Boucle For **: L3.** On parcourt les lignes caractérisées par leurs numéros i allant de 1 à n.

L4. On initialise sum à 0 : il s'agira du terme de droite

de l'inégalité (i fixé).

Boucle For : **L5.** On parcourt les colonnes caractérisées par leurs numéros allant de 1 à n.

=> Au final on parcourt les termes de la ligne i

L6. S'il ne s'agit pas d'un terme de la diagonale, i.e. si j n'est pas égal à i (numéro de ligne différent du numéro de colonne),

L7. On rajoute à sum la valeur absolue de ce terme.

L8. Ici, on conclut à propos de la vérification de l'inégalité pour une valeur de i. Si sum est supérieur à la valeur absolue du terme se positionnant à la fois sur la ligne i et sur la diagonale :

L9. Alors l'inégalité n'est pas vérifiée pour ce i : la matrice C n'est donc pas à diagonale dominante. On renvoie False.

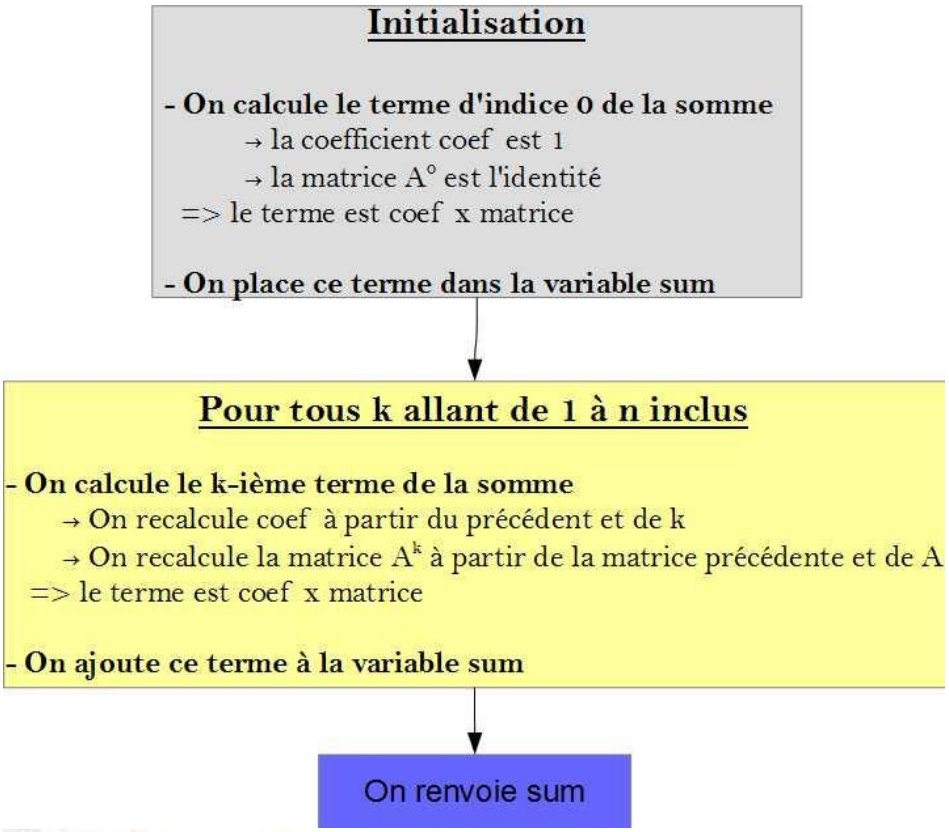
L10. Si on est arrivé ici, c'est qu'on n'a jamais renvoyé False et qu'on a vérifié l'inégalité pour tout i. Donc la matrice est à diagonale dominante. On renvoie True.

Exercice *M2/ ★ L'exercice consiste à créer une fonction **exp** non récursive qui a pour paramètres *une matrice A carrée* et *un entier k* et qui renvoie l'exponentielle de A à l'ordre k. Correction

L'exercice consiste à créer une fonction **exp** non récursive qui a pour paramètres *une matrice A carrée* et *un entier n* et qui renvoie l'exponentielle de A à l'ordre n.

ATTENTION : La notation k est devenue n pour plus de simplicité.

Code copiable : [Afficher]



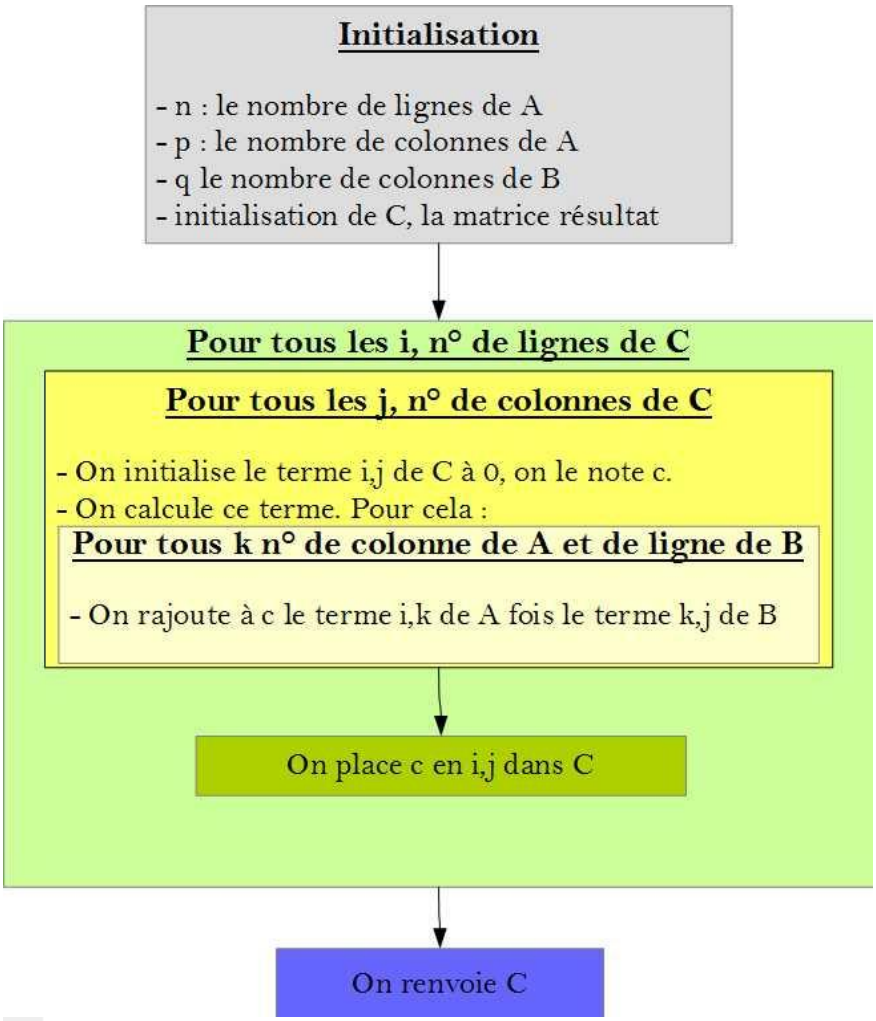

```python
import numpy as np

def exp(A,n):
    coef = 1
    matrice = np.eye(A.shape[0])
    sum = coef*matrice
    for k in range(1,n+1):
        coef = coef/k
        matrice=np.dot(matrice,A)
        sum += coef*matrice
    return(sum)
```

Exercice M3/ ★☆ L'exercice consiste à créer une fonction **expR** récursive qui a pour paramètres *une matrice A* et *un entier k* et qui renvoie l'exponentielle de A à l'ordre k.
(cf définition à l'exercice M2)

Exercice *M4/ ☆ L'exercice consiste à créer une fonction **mult** qui a pour paramètres *deux matrices A et B* et qui renvoie le produit matriciel AB (sans utiliser la fonction dot(A,B) qui renvoie déjà AB).

Correction

Code copiable : [Afficher]




```
1  import numpy as np
2
3  def mult(A,B):
4          n,p = A.shape
5          q = B.shape[1]
6          C = np.zeros((n,q))
7          for i in range(n):
8              for j in range(q):
9                  c = 0
10                 for k in range(p):
11                     c += A[i][k]*B[k][j]
12                 C[i,j] = c
13         return(C)
```

L1. On importe numpy car cette bibliothèque nous servira à créer la matrice résultant du produit de A par B.

L3. On commence la définition de fonction en donnant un nom à la fonction et à ses deux paramètres.

L4. On récupère le nombre de lignes n et de colonnes p de A à l'aide de la méthode shape qui renvoie le tuple (n,p).

L5. On récupère le nombre de colonnes q de B.

L6. On initialise la matrice résultant du produit de A par B à une matrice nulle à l'aide de la méthode zeros.

==> C est de taille n x q car elle résulte du produit d'une matrice de taille n x p et d'une matrice de taille p x q.

Boucle For : **L7.** On caractérise les lignes de C par un numéro i. Pour tout i :

Boucle For : **L8.** On caractérise les colonnes de C par un numéro j. Pour tout j :

L9. On initialise le terme i,j de c à 0.

Boucle For : **L10.** Pour tout k (nombre caractérisant le numéro de la colonne dans laquelle on se place dans A, et la ligne dans laquelle on se place dans B)

L11. On rajoute à c le terme i,k de A fois le terme k,j de B.

Fin Boucle For (sur k) **L12.** On place c en i,j dans C.

Fin Boucle For (x2) **L13.** On renvoie la matrice C.

Exercice M5/ ★ L'exercice consiste à créer une fonction **sym** qui a pour paramètre *une matrice carrée A* et qui renvoie True si la matrice est symétrique et False sinon.

DÉFINITION symétrique :[Afficher]

Exercice *M6/ - L'exercice consiste à créer une fonction **pdtvecto** qui a pour paramètres *deux vecteurs X et Y à 3 coordonnées*, et qui renvoie **le produit vectoriel de X avec Y.**

Correction

Code copiable : [Afficher]

```
import numpy as np

def pdtvecto(X,Y):
    Z0 = X[1,0]*Y[2,0] - X[2,0]*Y[1,0]
    Z1 = X[2,0]*Y[0,0] - X[0,0]*Y[2,0]
    Z2 = X[0,0]*Y[1,0] - X[1,0]*Y[0,0]
    return( np.array([[Z0,Z1,Z2]]) )
```

Attention ! Cette correction considère des vecteurs de type array([[a],[b],[c]]) de taille (3,1).

Attention aux indices !

Exercice *M7/ - Un chevalier et un martien vont au supermarché trois fois dans la semaine. Leurs nourrices voudraient savoir s'ils font attention aux prix.
A l'aide de la fonction **np.linalg.solve**, retrouve le prix de ce qu'ils ont acheté en connaissance de ce tableau : Correction

	Sabres achetés	**Paquets de chips achetés**	**Paquets de craies achetés**	**dépenses totale**
1ère fois	1	3	2	15 €
2ième fois	2	1	1	19 €
3ième fois	5	2	3	48 €

Code copiable : [Afficher]

```
1 import numpy as np
2
3 A = np.array([[1,3,2],[2,1,1],[5,2,3]])
4 B = np.array([[15],[19],[48]])
5
6 print(np.linalg.solve(A,B))
```

On a une équation matricielle AX = B
avec A la matrice des nombres d'objets achetés à chaque fois, X leur prix individuel et B les dépenses totales.

np.linalg.solve(A,B) nous renvoie X, correspondant au vecteur colonne composé de 8,1 et 2.
Un sabre vaut donc 8€, un paquet de chips en vaut 1, et un paquet de craies en vaut 2.

Exercice *M8/ ☆ Le tableau ci-dessous donne la hauteur de pins blancs en fonction de leurs diamètres. Réécrivez ce tableau dans un objet numpy.array et affichez :
- La hauteur moyenne,
- Le diamètre moyen,
- Le diamètre de l'arbre le plus grand.

Correction

Hauteur	127	119	135	132	130	130	110	75	110	124
Diamètre	21.2	20.2	24.6	23	27.2	18.6	17.3	10	19.7	22.3

Code copiable : [Afficher]

```
1 import numpy as np
2
3 T = np.array([[127,119,135,132,130,130,110,75,110,124],
4               [21.2,20.2,24.6,23,27.2,19.6,17.3,10,19.7,22.3]])
5
6 print("la hauteur moyenne est ", np.mean(T[0,:]))
7 print("le diamètre moyen est ", np.mean(T[1,:]))
8
9 i = np.argmax(T[0,:])
10 print("le diamètre de l'arbre le plus haut est ", T[1,i])
```

L1. On importe numpy renommé np.

L3. On réécrit le tableau en un array, en gardant la même structure : la première ligne correspond encore à la hauteur et la seconde correspond encore au diamètre.
L6. On affiche à l'aide de la fonction print. La moyenne est faite à l'aide de la fonction **mean**.
Pour la hauteur moyenne on ne récupère que la première ligne, avec toutes les colonnes : T[0, :]
L7. Pour le diamètre moyen on ne récupère que la seconde ligne, avec toutes les colonnes : T[1, :]

L9. Pour le diamètre de l'arbre le plus haut, on commence par récupérer i qui est l'indice de la colonne où la hauteur est maximale, à l'aide de **argmax**(T[0, :]) avec T[0, :] car on ne s'intéresse qu'aux valeurs de hauteur.
L10. On affiche le diamètre se situant à l'indice i, donc T[1,i].