Conseils :

- Prenez le temps de bien faire les choses et rendez une copie propre. Des points seront mis pour le soin (environ 1,5 points sur 20). Le fait de commenter ses fonctions pour les expliquer sera aussi évalué.
- Aucun ordre n'est imposé pour la résolution, par contre, rendez les problèmes dans l'ordre. Numérotez avec rigueur les questions que vous traitez.
- Même si vous n'avez pas sû faire une question, vous pouvez utiliser la fonction que vous étiez censé coder pour répondre aux questions suivantes.
- L'usage des **calculatrices est interdit**.

L'usage de fonctions de python non vues en cours telles que min/max/sum n'est pas autorisé sauf mention contraire.

I. Quelques petites fonctions

Les algorithmes seront écrits en python en respectant la syntaxe. L'algorithme devra être compréhensible par le lecteur. Chaque question demande l'écriture d'une fonction. Il est possible, de réutiliser une fonction définie à une question dans les questions suivantes. Les listes passées en argument des fonctions devront ne pas être modifiées par les fonctions.

Q1 ➢ Écrire une fonction `v_abs` qui prend en argument un nombre x et renvoie sa valeur absolue.

Q2 ➢ Programmer une fonction `f_conditionnel` correspondant à la fonction mathématique :

$$f : x \mapsto \begin{cases} 4 & \text{si } x < 0 \\ 5 & \text{si } x = 0 \\ -x^{100} & \text{si } x \in]0;3] \\ -\sqrt{x} & \text{si } x > 3 \end{cases}$$

Q3 ➢ Écrire une fonction `multiple` qui prend comme argument un nombre entier et renvoie `True` s'il est multiple de 13 et `False` sinon.

Q4 ➢ Écrire une fonction `mini2` qui prend comme argument deux nombres et qui renvoie la valeur du plus petit. En cas d'égalité le minimum est la valeur de n'importe lequel (puisque c'est la même). Par exemple `mini2(3,7)->3` et `mini2(7,7)->7`.

Q5 ➢ Écrire une fonction `mini4` qui prend comme argument quatre nombres et qui renvoie la valeur du plus petit. En cas d'égalité le minimum est la valeur de n'importe lequel (puisque c'est la même). Le code proposé ne devra pas prendre plus de 4 lignes.

Q6 ➢ Écrire une fonction `somme` qui prend comme argument une liste de nombres non vide et qui renvoie la somme des éléments de la liste.

Q7 ➢ Écrire une fonction `moyenne` qui prend comme argument une liste de nombres non vide et qui renvoie la valeur moyenne de la liste.

Q8 ➢ Écrire une fonction `somme_cos(n:int)->float` prenant en argument un entier n et renvoyant la somme $\sum_{k=0}^{n-1} \cos(2\pi \times k/n)$. On pourra importer la fonction cosinus et le nombre π de la bibliothèque math (appelés *cos* et *pi*).

Q9 ➢ Écrire une fonction `factorielle(n:int)->int` prenant en argument un entier n et renvoyant sa factorielle ($n!$). Par exemple `factorielle(5)->120`.

Q10 ➢ On considère la suite définie par la relation de récurrence $u_{n+1} = \frac{1}{0{,}1+|u_n|}$ et son premier terme u_0. Sans utiliser les listes, écrire une fonction `calcule_u(u0:float,n:int)->float` qui prend en argument le premier terme de la suite en un entier n et qui renvoie la valeur de u_n correspondante.

Q11 ➢ On considère la suite définie par $F_0 = 0$, $F_1 = 1$ et $F_n = 2 \times (F_{n-1})^2 + F_{n-2}$. Écrire une fonction `suite2(n:int)->int` qui prend en argument un entier n et qui renvoie la valeur de F_n. Ici aussi on évitera d'utiliser les listes.

➢ Bob et Alice se disputent : ils ne sont pas d'accord sur la manière de faire référence aux différents éléments d'un tableau.

Bob choisi de numéroter les cases en allant de gauche à droite, puis à chaque fois qu'il arrive au bout d'une ligne, il revient au début de la ligne suivante en continuant de compter (en gris sur la figure ci-dessous). À l'inverse, Alice trouve plus simple de faire référence à un élément en donnant son numéro de ligne et de colonne sous la forme d'un couple (ligne, colonne) (en noir sur la figure ci-contre). Nos deux amis choisissent de numéroter à partir de 0. Il y a en tout 7 lignes et 4 colonnes. Ainsi le premier élément du tableau est l'élément n°0 pour Bob et le (0,0) pour Alice. Le dernier est le n°27 pour Bob et le (6,3) pour Alice.

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)
16 (4,0)	17 (4,1)	18 (4,2)	19 (4,3)
20 (5,0)	21 (5,1)	22 (5,2)	23 (5,3)
24 (6,0)	25 (6,1)	26 (6,2)	27 (6,3)

Q12 Afin de réconcilier Bob et Alice, écrire une fonction `Alice_to_Bob(ligne:int, col:int)->int` permettant de réaliser la conversion de la numérotation d'Alice vers celle de Bob et une fonction `Bob_to_Alice(num:int)->(int,int)` permettant de faire l'inverse.

II. Simulation de la propagation d'une épidémie

Dans ce problème, on se propose d'étudier la propagation d'une épidémie. À la fin du problème, on obtiendra une fonction qui permet de simuler l'évolution du nombre de malades au sein d'une population. Pour cela, les questions feront coder petit à petit des «briques» de code sous forme de fonction. Si vous ne parvenez pas à coder une fonction à une question, il est néanmoins possible de l'utiliser dans la suite du problème à condition de respecter ses spécifications, c'est-à-dire :

- ➢ quels sont les paramètres en arguments et leur «type» (entier, liste etc...) ;
- ➢ quels sont les valeurs renvoyées et leur type ;
- ➢ quel est le comportement attendu de la fonction.

Vous veillerez à commenter vos programme afin d'expliquer «pourquoi/en quoi» ils ont le comportement attendu.

1 Description du modèle utilisé

On considère une population de N habitants, chaque habitant peut avoir 4 «états» par rapport à la maladie, chaque état sera représenté par un nombre entier dans notre programme :

- ➢ état «sain» : la maladie n'a pas été rencontré, cet état sera représenté par le nombre entier 0 ;
- ➢ état «incubation» : la personne était saine, mais a été contaminé, toutefois, elle n'a pas encore de symptômes et n'est pas contagieuse, représenté par le nombre 1 ;
- ➢ état «malade» : à l'issu de la période d'incubation, la personne a des symptômes et est potentiellement contagieuse, représenté par 2 ;
- ➢ état «guéri» : à l'issu de la maladie, la personne a guéri et est maintenant immunisée à la maladie, représenté par 3.

L'état de l'ensemble de la population sera représenté par une liste *etat* de N termes, telle que $etat[i]$ est l'état de la i-ème personne ($i \in [\![0, N-1]\!]$). Par exemple, si $etat$ = [1,1,0,2,3,0,1,2], alors il y a 8 personnes dans la population considérée. La personne numéro 0 est en train d'incuber la maladie, la numéro 2 est saine et la personne numéro 4 a déjà guéri.

On considérera un temps d'incubation de d_i jours et la durée de la phase «malade» sera de d_m jours. La probabilité qu'un malade risque de contaminer une personne chaque jour sera noté p.

La «simulation» consistera à faire évoluer la liste *etat* jour par jour. Ce modèle est simpliste mais peu servir de première base pour étudier l'influence des différents paramètres.

On donne la fonction *alea* qui pourra être utilisée (inutile de la ré-écrire).

```
import random
def alea(p):
    #renvoie True avec une probabilité p (et False avec proba 1-p)
    if random.random()<p:
        return True
    else:
        return False
```

2 Premières fonctions

Q13 ➢ Si $etat$ = [1,1,0,2,3,0,1,2], combien y a-t-il de personnes dans l'état «malade» ?

Q14 ➢ Écrire une fonction `nombre_malade(etat:list)->int` qui prend en argument une liste contenant l'état de la population et qui renvoie le nombre de personnes malades. Bien entendu, la fonction ne se limitera pas au cas de l'exemple précédent et devra être capable de s'adapter quelque soit la taille de la liste *etat*.

➢ On considère la fonction suivante :

```
def fonction_mystere(etat):
    res = [0,0,0,0]
    for x in etat:
        #x =0,1,2 ou 3
        res[x] = res[x] + 1
    return res
```

Q15 Exécuter à la main cette fonction pour la liste etat = [1,1,0,2,3,0,1,2] ; on expliquera dans la copie en détail ce que fait la boucle lors des 3 premiers passages dans la boucle, puis on donnera le résultat sans détailler les passages suivant dans la boucle.

Q16 ➢ Écrire une fonction `liste_malade(etat:list)->list` qui prend un argument une liste *etat* et qui renvoie la liste des numéros des personnes malades. Par exemple `liste_malade([3,2,0,2])->[1,3]`.

3 Contaminations

La fonction *randint* du module random a pour documentation :

randint(a, b)
Renvoie un entier aléatoire dans l'intervalle [a, b], y compris les deux bornes.

Chaque jour, un malade «essaie» (involontairement) de contaminer une personne. Si cette personne n'est pas saine, alors elle est immunisée à la maladie (déjà guérie ou déjà contaminée). Si la personne est saine, elle a une probabilité $p \in [0,1]$ d'être contaminée.

On souhaite écrire à la question Q18 une fonction `contamine1(num, etat, p)` qui prend en argument un entier *num* qui représente le numéro d'une personne (que l'on suppose ici contagieuse), une liste *etat*, et une probabilité de contaminer p et qui tire au hasard une autre personne, c'est-à-dire un entier différent de num dans l'ensemble $[\![0, N-1]\!]$, et qui la contamine avec une probabilité p. La fonction renvoie la nouvelle liste etat.

Par exemple `contamine1(0,[2,0],1)` doit renvoyer `[2,1]` alors que `contamine1(0,[2,0],0)` doit renvoyer `[2,0]`.

Q17 ➢ Expliquer les exemples proposés, sont ils justes, pourquoi ? Que pourrait renvoyer `contamine1(0,[2,0,1,0],0.5)`, donner toutes les possibilités.

Pour tirer un nombre aléatoire différent de *num* dans $[\![0, N-1]\!]$, on propose la technique suivante : on tire un nombre $n2$ dans l'ensemble $[\![0, N-2]\!]$, puis si $n2$ est supérieur ou égal à *num*, alors on lui ajoute 1.

Q18 ➢ Écrire la fonction `contamine1(num, etat, p)`. On pourra utiliser `randint` et `alea`.

4 Incubation

Pour gérer la durée d'incubation et de maladie, on se propose d'utiliser une deuxième liste *duree* qui contiendra pour chaque personne le nombre de jours depuis qu'elle est dans «son» état. Par exemple si

$etat = [1, 1, 0, 2, 3, 0, 1, 2]$ et
$duree = [2, 3, 9, 1, 0, 9, 1, 2]$,
alors la personne 0 incube depuis 2 jours, la personne 2 est saine depuis 9 jours etc...

Q19 ➢ Comment modifier la fonction `contamine1` pour tenir compte de la liste *duree* ? On considérera que la fonction ainsi modifiée s'appelle `contamine2(num,etat,p,duree)` et qu'elle renvoie à la fois la liste *etat*, mais aussi la liste *duree*. Vous n'êtes pas obligé de tout ré-écrire, mais il faudra indiquer clairement quelles modifications doivent être effectuées et où elles doivent l'être.

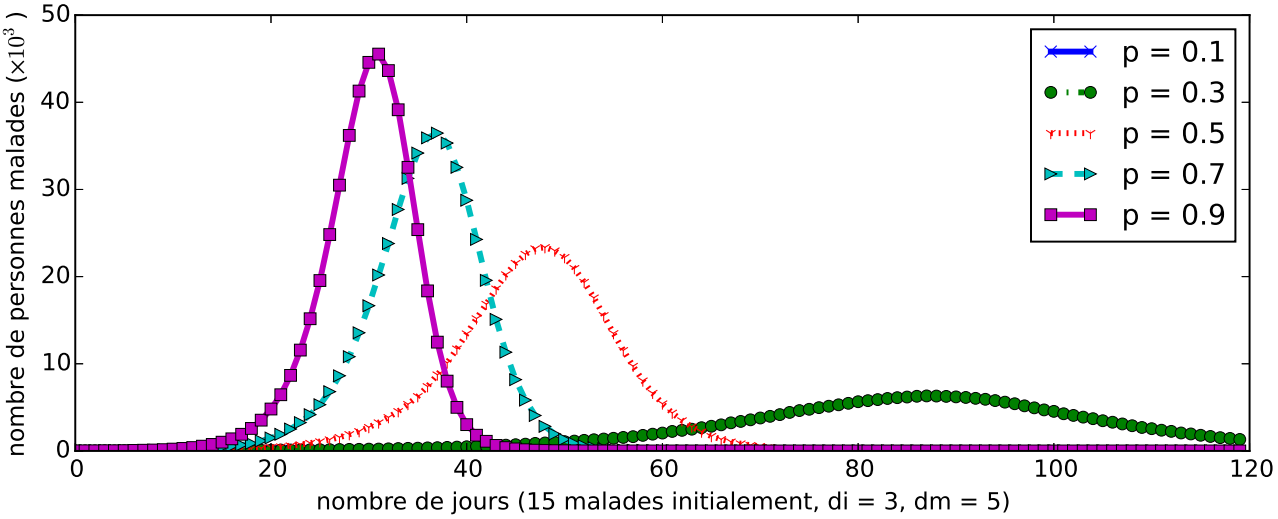
Q20 ➢ Écrire une fonction `contamine_all(etat,p,duree)` qui prend en argument la liste état, la probabilité p, et la liste duree et qui utilise la fonction `contamine2` pour chaque malade de la liste état, la fonction doit renvoyer les listes *etat* et *duree* après modification. On pensera à ré-utiliser judicieusement les fonctions déjà codée.

5 Plusieurs jours

Q21 ➢ Écrire une fonction `jour_suivant(etat,duree,p,d_i,d_m)` qui :
- prend en argument les listes *etat* et *duree* au début d'une journée, la probabilité p de contaminer quelqu'un, les durées d_i et d_m d'incubation et de maladie
- Ajoute 1 à la durée de chaque état et gère les passages de l'état 1 à l'état 2 lorsque la durée passée dans l'état 1 dépasse le temps d'incubation et de même pour de l'état 2 à l'état 3, puis réalise l'étape de contamination
- et enfin qui renvoie les listes etats et duree (qui correspondent donc à l'état de la population le lendemain matin).

Q22 ➢ Écrire une fonction `simule(N,p,d_i,d_m,Nb_jours,Nb_malade_init)` qui :
- prend en arguments N le nombre de personnes, p la probabilité de contamination, d_i et d_m les durées d'incubation et de maladie, *Nb_jours* la durée de la simulation et *Nb_malade_init* le nombre de malades le premier jour.
- la fonction doit initialiser les listes *duree* et *etat*, on fera en sorte que tous les malades soient au début de la liste *etat* pour plus de simplicité. Puis exécute la simulation *Nb_jours* fois en enregitrant le nombre de malades dans une liste. Par exemple *liste_nombre_malade*[i] contiendra le nombre de malades à la fin du jour i.
- et renvoie la liste *liste_nombre_malade*.

On pourrait ensuite tracer les courbes en changeant les paramètres et améliorer le modèle pour être plus fidèle à la réalité. Si vous vous ennuyez parce que le sujet est trop court, proposez des pistes d'amélioration ! (ci-dessous, où est la courbe $p = 0.1$?)



I. Quelques petites fonctions

Q1 ➢ Fonction vue en TP. Inutile de gérer le cas 0 à part puisque la fonction est continue. Ne pas faire $\sqrt{x^2}$, c'est mathématiquement juste, mais pensez à ce que ça vous couterait de calculer ainsi $|12345|$. D'ailleurs, essayer en python de calculer ainsi $|1.0 * 10 * *300|$ (n'oubliez pas le ".0"). Que répond python ?

```
1 def v_abs(x):
2     if x<0:
3         return -x
4     return x
```

Q2 ➢ Il s'agissait ici de voir si vous savez utiliser la structure if/elif/else. Attention au == et évitez de faire des tests « redondant » dans l'idéal (inutile de tester si $x > 0$ dans le 3e cas, ligne 6). Écrivez bien dans votre code `x**100` et non x^{100}.

```
1 def f_conditionnel(x):
2     if x<0:
3         return 4
4     elif x == 0:# == et non =
5         return 5
6     elif x<=3:#on sait que x>0 et x!=0, on
        est donc dans ]0, 3]
7         return -x**100
8     else: # x>0 et x!=0 et x> 3, donc x>3
9         return -x**0.5
```

Q3 ➢ Il s'agit ici de voir si vous pensez à utiliser l'opérateur modulo (%), qui est la manière usuelle de tester si un nombre divise un autre. On peut faire un if, mais ce n'est pas obligatoire (c'est plus « élégant » sans).

```
1 def multiple(n:int)->bool:
2     return n%13 == 0
3 #ou, mais un peu moins "élégant" :
4 def multiple(n:int)->bool:
5     if n%13 == 0:
6         return True
7     return False
```

Attention à ne pas mettre de guillemet, on voulait un booléen et non pas une chaîne de caractère.

Q4 ➢ Fonction très simple, mais il faut penser à la réutiliser à la question d'après. Comme pour la question précédente, le "else" n'est pas nécessaire car il y a des return dans toutes les conditions précédentes.

```
1 def mini2(a,b):
2     if a<b:
3         return a
4     return b # a ⩾ b
```

Q5 ➢ Réutiliser les fonctions précédentes est une bonne habitude à prendre. Évitez de refaire plusieurs fois le même calcul pour rien.

```
1 def mini4(a,b,c,d):
2     return mini2( mini2(a,b), mini2(c,d))
```

Q6 ➢ Fonction très classique, vue en TP et à connaitre sur le bout des doigts. La syntaxe `s += x` est équivalente à `s = s + x`. Autre possibilité ci-dessous.

```
1 def somme(L):
2     s = 0
3     for x in L:
4         s += x # ou s = s + x
5     return s
```

Attention au choix des noms de variables : même si on est en théorie libre, i, j, k évoque des entiers qui servent d'indice alors que $x, elt, valeur$ évoque des éléments de la liste. Ne pas respecter ces conventions n'est pas faux, mais risque de provoquer des confusions.

```
def somme(L):
    s = 0
    for i in range(len(L)):
        s += L[i] #s = s + L[i]
    return s
```

Q7 ➢ Fonction très classique, vue en TP et à connaître sur le bout des doigts.

```
def moyenne(L):
    return somme(L)/len(L)
```

Q8 ➢ Fonction vue en TP (qui devrait renvoyer 0 exactement s'il n'y avait pas d'erreur d'arrondi). Pour l'importation de cos et π, évitez d'utiliser `from math import *` et préférez `from math import pi,cos` ou `import math as m`.

```
import math as m
def somme_cos(n:int)->float:
    s = 0
    for i in range(n):
        s += m.cos(2*m.pi*i/n)
    return s
```

Q9 ➢ Fonction vue en TP. Attention à l'initialisation du produit qui est de 1 et non de 0. Attention à la boucle for qui doit aller de 1 à n inclus et non de 0 à $n-1$. On peut soit faire `range(1,n+1)`, soit écrire $i+1$ au lieu de i dans la boucle.

```
def factorielle(n:int)->int:
    p = 1
    for i in range(n):
        p = p * (i+1) # i+1 et non i
    return p
```

Q10 ➢ Calcul de suite récurrente d'ordre 1, assez classique. À savoir faire car cela peut reservir dans de nombreux cas. On peut réutiliser `v_abs`.

```
def calcule_u(u0:float , n:int)->float:
    for i in range(n):
        u0 = 1/(0.1+v_abs(u0))
    return u0
```

On fait bien range(n) qui s'exécute n fois, ainsi si $n = 1$, on passe une fois dans la boucle, on fait donc une fois le calcul qui permet de passer de u_0 à u_1, si on met $n = 2$, on passera bien à u_2 etc... Ainsi à la fin on a bien u_n.

➢ Calcul de suite récurrente d'ordre 2, assez classique mais nettement plus difficile que le précédent. À savoir faire car cela peut resservir dans de nombreux cas. Attention à gérer les cas de bases (F_0 et F_1) et à passer le bon nombre de fois dans la boucle (peut varier avec la structure exacte de votre programme). Ici, le nombre de fois que l'on passe dans la boucle mérite d'être

Q11 justifié car il est facile de se tromper.

```
def suite2(n:int)->int:
    if n == 0:
        return 0
    F0 = 0 #u_{n-2}
    F1 = 1 #u_{n-1}
    for i in range(n-1): # cf plus bas pour justification
        Fold = F0 # sauvegarde de l'ancien avant modif
        F0 = F1  # mise à jour de u_{n-1}
        F1 = 2*F0**2 + Fold# u_n = 2u_{n-1}^2+u_{n-2}
```

```
    #après 0 passage dans la boucle, on aurait F1
    #      1                                   F2
    #      n                                   Fn+1
    #      n-1                                 Fn
    return F1
```

> Dans beaucoup de cas, partez du principe que ce que l'on met dans le range est difficile et doit être justifié et vérifié.

Le seul cas facile est celui où l'on veut parcourir toute une liste ou faire quelque chose exactement n fois (`range(len(L))` ou `range(n)`).
Il faut en particulier vérifier que :

— on ne « sort pas de la liste », en particulier si dans la boucle on a des indices du type `L[i+1]` ou `L[i-1]` ;
— la boucle s'exécute le bon nombre de fois si $n = 0$ ou 1 ou 2 (voir s'il n'y a pas de décalage par exemple).

Autre possibilité :

```
def suite2(n:int)->int:
    F0 = 0 #u_{n}
    F1 = 1 #u_{n+1}
    for i in range(n): #n et non n-1
        Fold = F0
        F0 = F1  # mise à jour de u_{n}
        F1 = Fold + 2*F0**2 # u_{n+2} = u_{n}+2u_{n+1}^2
    return F0 # mais on renvoie F0 et non F1. On a calculé un
       terme pour rien
```

Q12 ➢ Conversion d'indice de tableau 1D/2D. Peut servir en informatique. Il faut penser à utiliser les modulos et divisions entières. Vu en TP !

```
def Alice_to_Bob(ligne, col):
    return ligne*4+col
#Pour chaque ligne en entier au
   dessus, il y a 4 numéro,
  puis dans la ligne courante,
   il faut ajouter le numéro
  de la colonne.
```

```
def Bob_to_Alice(num):
    return num//4, num%4
#le nombre de ligne est le
  nombre de fois où on a mis 4
   en entier, le numéro de la
  colonne est ce qui "reste"
  après.
```

II. Simulation de la propagation d'une épidémie

1 Description du modèle utilisé

L'énoncé ne pose pas de question, masi n'hésitez pas à vous en poser, à vérifier les exemples donnés ou à proposer un autre exemple. Cela permet de vérifier que vous avez compris et aussi de mieux mettre en mémoire les conventions de l'énoncés.

2 Premières fonctions

Q13 ➢ Si *etat* = [1,1,0,2,3,0,1,2], il y a deux malades (car on compte deux fois le nombre 2) dans la liste. Question très facile, mais qui est là pour vous aider à vous approprier l'énoncé et qui prépare la question suivante. C'est typiquement une question qu'il aurait fallu vous poser avant de faire la fonction de la question suivante.

Q14 ➢ Fonction `nombre_malade(etat:list)->int` : compte tenu de ce que l'on a dit à la question précédente, il suffit de compter le nombre de 2 dans la liste état.

```
def nombre_malade(etat):
    n = 0 #nombre de malade
    for x in etat: #pour chaque personne
        if x == 2: #si l'état de la personne (x) est malade (2)
            n+= 1 #on ajoute 1 au nombre de malades trouvés
    return n #attention à l'indentation
```

➢ On considère la fonction suivante :

```
def fonction_mystere(etat):
    res = [0,0,0,0]
    for x in etat:
        #x =0,1,2 ou 3
        res[x] = res[x] + 1
    return res
```

Q15 Exécutons à la main cette fonction pour la liste etat = [1,1,0,2,3,0,1,2] ;

1. Première itération x prend la valeur 1, on augmente donc la valeur de $res[1]$ de 1 et donc $res = [0,1,0,0]$;
2. Deuxième itération x prend la valeur 1, on augmente donc la valeur de $res[1]$ de 1 et donc $res = [0,2,0,0]$;
3. Troisième itération x prend la valeur 0, on augmente donc la valeur de $res[0]$ de 1 et donc $res = [1,2,0,0]$;

À la fin, $res = [2,3,2,1]$; c'est-à-dire $res[0]$ contient le nombre de fois où l'on a rencontré 0 dans la liste *etat*, de même $res[1]$ contient le nombre d'occurence de 1, $res[2]$ le nombre de 2, et $res[3]$ le nombre de 3.

Q16 ➢ Fonction `liste_malade(etat:list)->list` qui prend un argument une liste *etat* et qui renvoie la liste des numéros des personnes malades. Ici, un peu comme précédemment on cherche les 2 dans la liste *etat*, mais cette fois on veut "sauver" leurs indices dans une liste.

```
def liste_malade(etat):
    res = [] #le résultat sera une liste de taille inconnue
```

```
    for i in range(len(etat)): #syntaxe avec i in range car on a
        besoin des indices
        if etat[i] == 2: #si la i-ème personne est malade
            res.append(i) #on enregistre son indice
    return res
```

3 Contaminations

Par exemple contamine1(0,[2,0],1) doit renvoyer [2,1] alors que contamine1(0,[2,0],0) doit renvoyer [2,0].

➢ Encore une fois, la question qui est posée ici est une question qu'il faudrait se poser soi-même si l'énoncé ne nous l'avait pas demandé, et ce afin de mieux comprendre l'énoncé. `contamine1(0,[2,0],1)`, la personne d'indice 0 (qui est effectivement malade (2) d'après la liste *etat*) "essaye" de contaminer quelqu'un, la seule autre personne (la numéro 1) est "saine"(0) et peut donc bien tomber malade. De plus, la probabilité de contamination est de 1, c'est à dire 100%, et donc la personne 1 va contracter la maladie et donc commencer à incuber, la liste état devient donc $[2, 1]$. Dans le 2e exemple, tout est pareil, sauf la probabilité de contamination qui est nulle et donc la personne 1 ne sera pas contaminée et la liste état devient donc [2,0].

Q17 Que pourrait renvoyer *contamine*1(0,[2,0,1,0],0.5) ? La personne 0 peut essayer de contaminer la personne 1, ou la 2, ou la 3. La 2 ne peut pas être contaminée étant déjà malade, la 1 et la 3 peuvent être contaminée, mais la probabilité est de 1/2. Les réponses possibles sont :

- [2,0,1,0] si c'est la personne 2 que l'on a essayé de contaminer ou si la contamination a échouée ;
- [2,1,1,0] si c'est la personne 1 que l'on a essayé de contaminer et que la contamination a réussi ;
- [2,0,1,1] si c'est la personne 3 que l'on a essayé de contaminer et que la contamination a réussi.

Q18 ➢ Fonction `contamine1(num, etat, p)`.

```
def contamine1(num,etat,p):
    """ étape de contamination si la personne num est malade : 
       on essaye de contaminer une AUTRE personne au hasard, si 
       la personne n'est pas saine, cela échoue, sinon, cela 
       réussit avec une probabilité p, c'est-à-dire que l'autre 
       personne se met à incuber """
    N = len(etat)
    num2 = random.randint(0,N-2) #N-1 autres personnes
    if num2>= num:
        num2 += 1 #de façon à tirer un nombre aléa entre 0 et N
           -1, mais qui exclu num parce qu'on va pas se
           contaminer soi-même
    #num2 est la personne que l'on va essayer de contaminer,
       mais on ne peut la contaminer qui si elle est saine
    if etat[num2] == 0 and alea(p): #si la personne num2 est
       saine et avec une proba p
        etat[num2] = 1 #elle incube, sinon, inutile de changer
           la liste etat
    return etat #en fait pas nécessaire parce que etat est une
       liste
```

4 Incubation

Q19 ➢ Comment modifier la fonction `contamine1` ? Il faut bien entendu changer le def : **def** `contamine2(num,etat,p,duree)` ; il faut changer le if en ajoutant $duree[num2] = 0$; et enfin changer le return pour rajouter *duree*.

```
def contamine2(num,etat,p,duree): #modif
    N = len(etat)
    num2 = random.randint(0,N-2)
    if num2>= num:
        num2 += 1
    if etat[num2] == 0 and alea(p):
        etat[num2] = 1
        duree[num2] = 0 #modif
    return etat,duree #modif
```

Q20 ➢ Écrire une fonction `contamine_all(etat,p,duree)` qui prend en argument la liste état, la probabilité p, et la liste duree et qui utilise la fonction `contamine2` pour chaque malade de la liste état, la fonction doit renvoyer les listes *etat* et *duree* après modification. On pensera à ré-utiliser judicieusement les fonctions déjà codée.+

```
def contamine_all(etat,p,duree):
    for num in liste_malade(etat): #on réutiliser la fonction
       liste malade pour trouver tous les malades
        etat,duree = contamine2(num, etat,p,duree) #on utilise
           la fonction précédente, attention à affecter le
           résutlat
    return etat,duree
```

5 Plusieurs jours

Q21 ➢ Écrire une fonction `jour_suivant(etat,duree,p,d_i,d_m)` :

```
def jour_suivant(etat,duree,p,d_i,d_m):
    N = len(etat)
    for num in range(N): #pour chaque personne
        duree[num] += 1 #on augmente la duree de l'état actuel
        #mais si on dépasse la durée max, on passe à l'état
           suivant
        if etat[num] == 1 and duree[num] >= d_i:
            etat[num] = 2 #etat suivant
            duree[num] = 0 #on réinitialise la durée puisque l'
               on a changé d'état
        elif etat[num] == 2 and duree[num] >= d_m:
            etat[num] = 3
            duree[num] = 0
    return contamine_all(etat,p,duree) #et finalement on
       contamine les autres
```

Q22 ➢ Écrire une fonction `simule(N,p,d_i,d_m,Nb_jours,Nb_malade_init)`

```
def simule(N,p,d_i,d_m,Nb_jours,Nb_malade_init):
    duree = [0] * N
```

```
    etat = [0] * N
    liste_nombre_malade = []

    for i in range(Nb_malade_init):
        etat[i] = 2
    for jour in range(Nb_jours):
        etat,duree = jour_suivant(etat, duree,p,d_i,d_m)
        liste_nombre_malade.append(nombre_malade(etat))

    #optionnel pour graphique avec import matplotlib.pyplot as
       plt
    plt.plot(liste_nombre_malade)
    #fin optionnel pour graphique
    return liste_nombre_malade
```