TD n°3 de probabilités : Arbres des probabilités

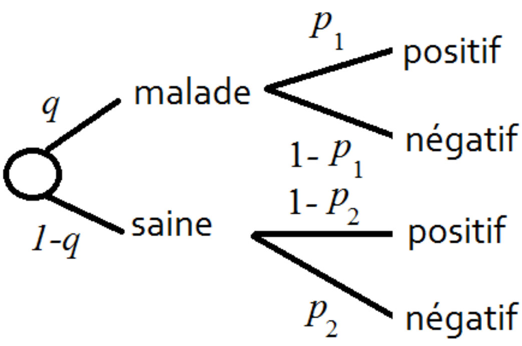
CORRECTION

1. Test biologique

On dispose d'un test pour détecter une maladie présente dans la population avec une fréquence q. Avec ce test, la probabilité d'obtenir un résultat positif sur une personne malade est p_1 tandis que la probabilité d'obtenir un résultat négatif sur une personne saine est p_2.

a) Dresser l'arbre des probabilités de la situation.
Déterminer les probabilités des 4 issues de cet arbre.

D'une façon générale, dépendant des paramètres q, p_1 et p_2, on peut dresser l'arbre des probabilités ci-contre :



Comme la probabilité d'un chemin est égale au produit des probabilités rencontrées sur le chemin, on a :

P(positif et malade)= $q\times p_1$;

P(négatif et malade)= $q\times(1-p_1)$;

P(positif et saine)= $(1-q)\times(1-p_2)$;

P(négatif et saine)= $(1-q)\times p_2$.

b) En déduire la probabilité P(pos) d'avoir un résultat positif

La probabilité d'un événement formé par la réunion de plusieurs chemins (les chemins sont, par nature, incompatibles entre eux) est égale à la somme des probabilités de ces chemins. Ainsi, on a :

P(positif)=P(positif et malade)+P(positif et saine)= $q\times p_1+(1-q)\times(1-p_2)$.

et la probabilité P(neg) d'avoir un résultat négatif

De même, on a P(négatif)=P(négatif et malade)+P(négatif et saine)= $q\times(1-p_1)+(1-q)\times p_2$.

Évidemment, on aurait pu remarquer que « négatif » est le contraire de « positif », et donc on aurait pu faire autrement : P(négatif)= 1− P(positif)= $1-(q\times p_1+(1-q)\times(1-p_2))=1-q\times p_1-(1-q)\times(1-p_2)$.

Est-ce la même chose ? Oui, car

$1-q\times p_1-(1-q)\times(1-p_2)=1-q\,p_1-(1-q)+p_2(1-q)=q-q\,p_1+p_2-q\,p_2=q\times(1-p_1)+(1-q)\times p_2$.

c) On veut déterminer la probabilité p_3 qu'une personne ayant obtenu un résultat positif soit saine (« faux positif ») et la probabilité p_4 qu'une personne ayant obtenu un résultat négatif soit malade (« faux négatif »). Pour cela, on peut supposer une population de 1000 personnes et déterminer la fréquence des personnes de chaque catégorie (malade-positif, malade- négatif, saine-positif, saine-négatif). Montrer finalement que : $p_3=\dfrac{(1-q)(1-p_2)}{(1-q)(1-p_2)+q\,p_1}$.

Sur 1000 personnes, les malade-positifs sont $1000\,q\,p_1$,
les malade-négatifs sont $1000\,q\,(1-p_1)$,
les saine-positifs sont $1000\,(1-q)(1-p_2)$ et
les saine-négatifs sont $1000\,(1-q)\,p_2$.
Il y a $1000\,q\times p_1+(1-q)(1-p_2)$ positifs parmi lesquels $1000\,(1-q)(1-p_2)$ sont saines. La probabilité cherchée est donc $p_3=\dfrac{1000\,(1-q)(1-p_2)}{1000\,(1-q)(1-p_2)+q\,p_1}$. Il ne reste plus qu'à simplifier par l'effectif de la population qui n'apporte rien au problème. La probabilité p_3 d'un « faux positif » est donc $p_3=\dfrac{(1-q)(1-p_2)}{(1-q)(1-p_2)+q\,p_1}$.

Établir une formule similaire pour p_4.

Pour calculer la probabilité p_4 d'un « faux négatif », on ne va pas utiliser l'effectif de la population (puisqu'il disparaît à la simplification finale). Il suffit de diviser la fréquence des cas favorables (personnes malade-négatives) par la fréquence des cas possibles (personnes négatives) : $p_4=\dfrac{q\,(1-p_1)}{q\,(1-p_1)+(1-q)\,p_2}$

d) Discuter l'efficacité du test après avoir calculé p_3 et p_4 dans les quatre cas suivants :

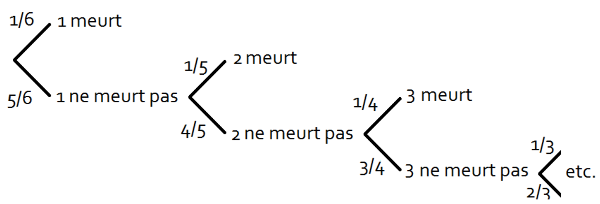
	q=0,1 – p_1=0,9 – p_2=0,9	q=0,1 – p_1=0,999 – p_2=0,9	q=0,1 – p_1=0,9 – p_2=0,999	q=0,01 – p_1=0,9 – p_2=0,999
p_3	0,5	0,474	0,010	0,099
p_4	0,01	0,00012	0,011	0,001

Dans le 1er cas, on voit que les faux positifs sont très nombreux (la moitié des personnes donnant un test

positifs ne sont pas malades, ce qui n'est pas très satisfaisant...). Augmenter p_1 (la détection des vrais malades) ne change pas beaucoup ce nombre : on le voit dans la 2$^{\text{ème}}$ colonne où la fréquence des faux positifs ne change pas vraiment. Pour faire chuter les faux positifs, il faut que le test ne détecte pas la maladie chez les personnes saines, et donc, il faut agir sur p_2 : lorsqu'une personne sur 1000 seulement est déclarée malade (positive) alors qu'elle est saine, la fréquence des faux-positifs atteint le seuil plus acceptable de 1%. Lorsque la maladie est dix fois plus rare (q=0,01 au lieu de 0,1), les faux-positifs remontent. Il faudrait, pour se maintenir au seuil des 1% de faux-positifs, augmenter la fiabilité du test sur les personnes saines (passer p_2 à une valeur supérieure : avec q=0,01 - p_1=0,9 - p_2=0,9999, on obtient $p_3=0{,}0011$, ce qui est encore un peu supérieur à 1%).
Le même genre de commentaire peut se faire sur p_4. Comme on le constate, les fréquences sont faibles (car il y a peu de malades et qu'on en détecte tout de même un grand nombre) mais l'enjeu pour les faux-négatifs est cependant plus grave que pour les faux-positifs car ce sont des personnes malades qui ne sont pas détectées par le test, et donc ce sont des personnes qui ne seront pas prises en charge pour un traitement médical ou un suivi.

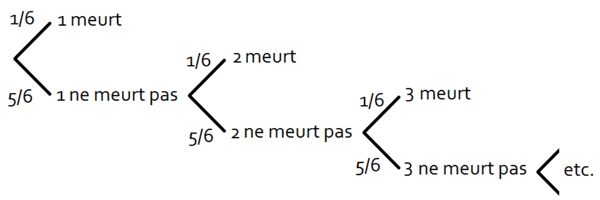
2. Roulette Russe

Imaginons qu'on dispose d'un revolver dont le barillet peut contenir 6 balles. On a mis une seule balle dans le barillet, puis on a fait tourner celui-ci pour ne pas savoir où est la balle. Chacune des 6 personnes assises autour de la table doit approcher le canon de sa tempe et tirer. Si la balle ne l'a pas tuée, la personne qui vient de tirer doit passer le revolver à son voisin.
On demande d'examiner la probabilité p_1 que ce soit la 1$^{\text{ère}}$ personne qui meure, p_2 que ce soit la 2$^{\text{de}}$, etc. dans les scénarios suivants. La question étant de déterminer quelle est la place la plus dangereuse à ce jeu stupide.



a) S_1 : on ne modifie rien au revolver à chaque passage

La 1$^{\text{ère}}$ personne a 1 « chance » sur 6 de mourir. Pour calculer la probabilité que ce soit la 2$^{\text{de}}$ qui meure, il faut considérer le résultat du 1$^{\text{er}}$ tir et celui du 2$^{\text{ème}}$.
Pour analyser la situation, on peut utiliser l'arbre des probabilités ci-contre. La probabilité que ce soit la 2$^{\text{de}}$ qui meure est donc de $\frac{5}{6}\times\frac{1}{5}=\frac{1}{6}\approx 0{,}167$, la probabilité que ce soit la 3$^{\text{ème}}$ qui meure est donc de $\frac{5}{6}\times\frac{4}{5}\times\frac{1}{4}=\frac{1}{6}\approx 0{,}167$, etc.
Dans tous les cas, à toutes les places, la probabilité de mourir est la même.
Ce jeu a beau être stupide, au moins, il est équitable.
Ce résultat semble paradoxal : la 6$^{\text{ème}}$ personne a une probabilité $p_6=\frac{5}{6}\times\frac{4}{5}\times\frac{3}{4}\times\frac{2}{3}\times\frac{1}{2}\times\frac{1}{1}=\frac{1}{6}\approx 0{,}167$ de mourir alors que, lorsque le revolver lui parvient, elle est certaine de mourir. Sauf qu'il ne s'agit pas des mêmes probabilités : avant que la 1$^{\text{ère}}$ personne ne tire, la probabilité que ce soit l'une ou l'autre est égale. Après ce tir, si la 1$^{\text{ère}}$ n'est pas morte, chacune des 5 personnes restantes a une probabilité de mourir égale à $\frac{1}{5}$ … etc. La probabilité que le 6$^{\text{ème}}$ joueur doive tirer à son tour est égale à celle de l'événement « les 5 premières personnes ne sont pas mortes » et vaut, comme on l'a vu, $\frac{1}{6}$.



b) S_2 : on tourne le barillet du revolver à chaque passage

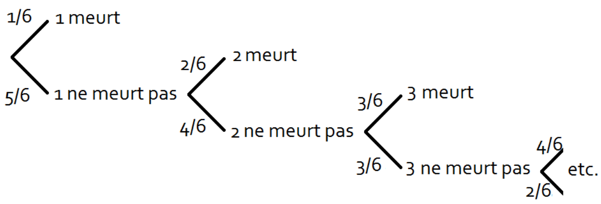
Ici, l'arbre des probabilités est encore plus simple. Le fait de tourner le barillet à chaque fois redonne une probabilité de 1 sur 6 de mourir. La probabilité que ce soit la 2$^{\text{de}}$ qui meure n'est donc plus que de $\frac{5}{6}\times\frac{1}{6}=\frac{5}{36}\approx 0{,}139$, etc. La place la plus dangereuse est donc la 1$^{\text{ère}}$, ensuite cela va en décroissant (en multipliant par $\frac{5}{6}$).
Il peut se trouver avec cette règle, que la 6$^{\text{ème}}$ personne ne meure pas et que l'on doive recommencer un 2$^{\text{ème}}$ tour... Cela va augmenter globalement le risque de mourir de chacun (ainsi le 1$^{\text{er}}$ aura une probabilité de mourir de 0,251 lorsqu'on

roulette russe, scénario n°2

1	2	3	4	5	6	
0,16667	0,13889	0,11574	0,09645	0,08038	0,06698	1$^{\text{er}}$ tour
0,05582	0,04651	0,03876	0,03230	0,02692	0,02243	2$^{\text{ème}}$ tour
0,01869	0,01558	0,01298	0,01082	0,00901	0,00751	3$^{\text{ème}}$ tour
0,00626	0,00522	0,00435	0,00362	0,00302	0,00252	4$^{\text{ème}}$ tour
0,00210	0,00175	0,00146	0,00121	0,00101	0,00084	5$^{\text{ème}}$ tour
0,00070	0,00059	0,00049	0,00041	0,00034	0,00028	6$^{\text{ème}}$ tour
0,00024	0,00020	0,00016	0,00014	0,00011	0,00009	7$^{\text{ème}}$ tour
0,00008	0,00007	0,00005	0,00005	0,00004	0,00003	8$^{\text{ème}}$ tour
0,00003	0,00002	0,00002	0,00002	0,00001	0,00001	9$^{\text{ème}}$ tour
0,00001	0,00001	0,00001	0,00001	0,00000	0,00000	10$^{\text{ème}}$ tour
0,251	0,209	0,174	0,145	0,121	0,101	total

tient compte des 10 tours, voir le tableau ci-contre), mais la 1ère place restera toujours la plus dangereuse. Quelle est la probabilité qu'aucune des 6 personnes ne meure dans le 1er tour et que l'on doive commencer un 2ème tour : $\left(\frac{5}{6}\right)^6 \approx 0{,}335$. On peut trouver ce nombre en effectuant la somme :
$1-\left(\frac{1}{6}+\frac{5}{36}+\frac{25}{216}+\frac{125}{1296}+\frac{625}{7776}+\frac{3125}{46656}\right) \approx 0{,}335$ puisqu'il s'agit du contraire de « l'un des 6 joueurs meurt au 1er tour ». Si on poursuit au delà du 1er tour, il y a une probabilité de 0,11216 d'aller au delà du 2ème. Pour parvenir à commencer le 10ème tour, il faut que personne ne meure avant, cet événement n'a une probabilité que de 0,00005 environ.

c) S_3 : on ajoute une balle dans le barillet et on tourne celui-ci à chaque passage



Dans ce scénario, on précipite la fin du jeu et le dernier est sûr de mourir si la « chance » épargne les 5 premiers, comme dans le 1er scénario, mais ce qui change est que la probabilité que cela arrive est bien moindre.
Calculons les probabilités à partir de l'arbre ci-contre qui résume la situation. La probabilité que ce soit la 2de personne qui meure est de $\frac{5}{6}\times\frac{2}{6}=\frac{10}{36}\approx 0{,}278$, la probabilité que ce soit la 3ème qui meure est de $\frac{5}{6}\times\frac{4}{6}\times\frac{3}{6}=\frac{60}{216}=\frac{10}{36}\approx 0{,}278$, la probabilité que ce soit la 4ème qui meure est de $\frac{5}{6}\times\frac{4}{6}\times\frac{3}{6}\times\frac{4}{6}=\frac{240}{1296}=\frac{5}{27}\approx 0{,}185$, etc. La probabilité que le 6ème joueur doive tirer à son tour est égale à celle de l'événement « les 5 premières personnes ne sont pas mortes » et vaut $\frac{5}{6}\times\frac{4}{6}\times\frac{3}{6}\times\frac{2}{6}\times\frac{1}{6}\times\frac{6}{6}=\frac{6!}{6^6}=\frac{720}{46656}=\frac{5}{324}\approx 0{,}015$.
Le tableau ci-dessous résume les trois scénarios.

	p_1	p_2	p_3	p_4	p_5	p_6
S_1	$\frac{1}{6}\approx 0{,}167$	$\frac{1}{6}\approx 0{,}167$	$\frac{1}{6}\approx 0{,}167$	$\frac{1}{6}\approx 0{,}167$	$\frac{1}{6}\approx 0{,}167$	$\frac{1}{6}\approx 0{,}167$
S_2	$\frac{1}{6}\approx 0{,}167$	$\frac{5}{36}\approx 0{,}139$	$\frac{25}{216}\approx 0{,}116$	$\frac{125}{1296}\approx 0{,}096$	$\frac{625}{7776}\approx 0{,}080$	$\frac{3125}{46656}\approx 0{,}067$
S_3	$\frac{1}{6}\approx 0{,}167$	$\frac{10}{36}\approx 0{,}278$	$\frac{10}{36}\approx 0{,}278$	$\frac{240}{1296}\approx 0{,}185$	$\frac{600}{7776}\approx 0{,}077$	$\frac{720}{46656}\approx 0{,}015$

Conclusion : quelle place est la plus dangereuse

avec S_1 ? Toutes les places sont aussi dangereuses. Personnellement si je devais jouer un jour à ce jeu stupide (cela n'arrivera jamais), et si j'avais la possibilité de choisir ma place, je choisirai la dernière mais pour les cardiaques, il vaut mieux choisir la 1ère place car après avoir tiré, s'ils ne sont pas morts, ils seront tranquilles.
avec S_2 ? Que l'on regarde le 1er tour uniquement ou que l'on considère les 10 premiers tours potentiels (voir mon tableau, plus haut), c'est la 1ère place la plus dangereuse, suivie de la 2de, etc.
avec S_3 ? Ce sont les 2ème et 3ème places *ex-aequo* les plus dangereuses, suivies de la 4ème. La 1ère place n'est classée que 4ème dans l'ordre de la dangerosité. La dernière place est alors la plus sûre (mais on y est aussi sûr de mourir, comme d'ailleurs dans le scénario 1) car dans 98,5% des cas le jeu s'arrêtera avant que cette 6ème personne doive jouer à son tour.

Prolongement : Écrire un algorithme qui simule le scénario S_3 de manière à reproduire n fois l'expérience. À chaque fois, on s'arrête et on note la place de la personne tuée. En prenant n suffisamment grand, on s'approche ainsi des probabilités cherchées.

On peut utiliser 6 mémoires (A, B, C, D, E et F) et incrémenter le contenu de la mémoire concernée mais il y a mieux : un tableau, une structure de mémoire qui possède un indice. Par exemple, si A est le nom d'un tableau de 6 nombres, ces nombres sont $A[0]$, $A[1]$, …$A[5]$. Si le joueur tué est p, on incrémente ainsi le nombre A[p], l'algorithme en est simplifié.

```
Lire N                                   N est le nombre de fois que l'on veut simuler cette expérience aléatoire
Initialiser les valeurs d'un tableau contenant 6 nombres (Pour J allant de 1 à 6 : A[J]=0)
Pour I allant de 1 à N
    B=0                  B vaut 0 tant que personne n'est mort, ensuite (lorsqu'un joueur meurt) B vaut 1
    R=1                                          R est le nombre de balles dans le Revolver
    J=1                                                J est la place du joueur qui tire
    Tant que B=0              En programmant ainsi la boucle, on l'arrête dès qu'une personne meure
        C=Ent(Random*6+1)       On tire un nombre entier au hasard entre 1 et 6 pour simuler le tir
        Si C≤R Alors {A[J]=A[J]+1 ; B=1}
            Sinon {J=J+1; R=R+1}    Cette dernière instruction (R=R+1) ajoute une balle dans le Revolver
Pour J allant de 1 à 6 : afficher A[J]/N                   La fréquence des morts à la place J
```

Voilà! Il ne reste plus qu'à programmer cet algorithme contenant deux boucles imbriquées l'une dans l'autre. C'est le tableau *A* qui est intéressant ici : une structure nouvelle de données qui vous sera très utile par la suite.

Commençons par utiliser Algobox pour tester notre algorithme est en donner des copies d'écran. Nous n'avons pas utilisé Algobox cette année mais je pense que vous pouvez voir que la programmation y est assez naturelle, un peu lourde mais claire et proche de l'algorithme.
Vous avez remarqué que l'initialisation du tableau demande une boucle spécifique (lignes 11 à 14 de notre programme). Certains langages de programmation supposent que les valeurs d'un tableau sont nulles au départ, mais ce n'est pas le cas pour Algobox qui refusera d'exécuter le programme sans cette initialisation.

```
1   VARIABLES
2     I EST_DU_TYPE NOMBRE
3     J EST_DU_TYPE NOMBRE
4     B EST_DU_TYPE NOMBRE
5     N EST_DU_TYPE NOMBRE
6     R EST_DU_TYPE NOMBRE
7     A EST_DU_TYPE LISTE
8     C EST_DU_TYPE NOMBRE
9   DEBUT_ALGORITHME
10    LIRE N
11    POUR J ALLANT_DE 1 A 6
12      DEBUT_POUR
13      A[J] PREND_LA_VALEUR 0
14      FIN_POUR
15    POUR I ALLANT_DE 1 A N
16      DEBUT_POUR
17      B PREND_LA_VALEUR 0
18      R PREND_LA_VALEUR 1
19      J PREND_LA_VALEUR 1
20      TANT_QUE (B==0) FAIRE
21        DEBUT_TANT_QUE
22        C PREND_LA_VALEUR 23   floor(random()*6+1)
24        SI (C<=R) ALORS
25          DEBUT_SI
26          A[J] PREND_LA_VALEUR A[J]+1
27          B PREND_LA_VALEUR 1
28          FIN_SI
29          SINON
30            DEBUT_SINON
31            J PREND LA VALEUR J+1
32            R PREND_LA_VALEUR R+1
33            FIN SINON
34        FIN TANT QUE
35      FIN_POUR
36    POUR J ALLANT_DE 1 A 6
37      DEBUT_POUR
38      AFFICHER "fréquence de mort à la "
39      AFFICHER J
40      SI (J==1) ALORS
41        DEBUT_SI
42        AFFICHER "ère place : "
43        FIN_SI
44        SINON
45          DEBUT SINON
46          AFFICHER "ème place : "
47          FIN_SINON
48      C PREND_LA_VALEUR A[J]/N*100
49      AFFICHER C
50      AFFICHER " %"
51      FIN_POUR
52    AFFICHER "Calculs faits avec "
53    AFFICHER N
54    AFFICHER " valeurs"
55  FIN ALGORITHME
```

Résultats

```
fréquence de mort à la 1ère place : 14 %
fréquence de mort à la 2ème place : 30 %
fréquence de mort à la 3ème place : 26 %
fréquence de mort à la 4ème place : 22 %
fréquence de mort à la 5ème place : 6 %
fréquence de mort à la 6ème place : 2 %
Calculs faits avec 100 valeurs
```

```
fréquence de mort à la 1ère place : 16.1 %
fréquence de mort à la 2ème place : 30 %
fréquence de mort à la 3ème place : 26.3 %
fréquence de mort à la 4ème place : 17.5 %
fréquence de mort à la 5ème place : 8.4 %
fréquence de mort à la 6ème place : 1.7 %
Calculs faits avec 1000 valeurs
```

```
fréquence de mort à la 1ère place : 16.34 %
fréquence de mort à la 2ème place : 27.8 %
fréquence de mort à la 3ème place : 28.41 %
fréquence de mort à la 4ème place : 18.5 %
fréquence de mort à la 5ème place : 7.4 %
fréquence de mort à la 6ème place : 1.55 %
Calculs faits avec 10000 valeurs
```

Outre le fait d'être un bon exercice d'algorithmique et une initiation aux tableaux, vous apprécierez de retrouver, ici, les valeurs prévues par notre calcul de probabilités. Regardez le résultat après N=10 000 simulations : 16,34% ; 27,8% ; 28,41% ; 18,5% ; 7,4% ; 1,55%
Comparer avec les valeurs réelles des probabilités : 16,67% ; 27,78% ; 27,78% ; 18,52% ; 7,72% ; 1,54%

Voyons maintenant la programmation de cet algorithme avec Python.
L'initialisation de la liste en Python peut se faire de plusieurs façons, nous avons choisi une déclaration explicite : A=[0,0,0,0,0,0].
On déclare que la variable A est un tableau (une liste) contenant 6 emplacements et on affecte 0 dans les 6 emplacements. A[0] indique le nombre de morts en position 1 ; A[1] le nombre de morts en position 2 ; etc.
Il y a un décalage entre l'indice i de A[i] et la position au jeu car les tableaux commencent toujours à l'indice 0 en Python. C'est pour cela que dans l'affichage final on met A[J] pour la fréquence de mort en J+1^ème^ place.
Notez comme c'est plus simple de programmer les boucles et les tests car il n'y a pas d'instruction de fin, juste une indentation et « : »
Je rappelle qu'en Python, ajouter 1 dans une variable A peut s'écrire : A+=1 mais on peut aussi utiliser la syntaxe A=A+1. Dans tous les cas l'égalité est une affectation : A=3 affecte 3 dans la variable A ; par contre, A==3 (avec deux signes =) est un test : A est-il égal à 3 ? Cela explique la boucle While B==0 qui signifie « tant que personne n'est mort » car B contient 0 au début jusqu'à ce que quelqu'un meure.

```
#scénario 3 à la roulette russe
from random import randint

N=int ( input ( "Combien d'essais? ") )
A=[0,0,0,0,0,0]
for I in range ( N ) :
    B,R,J=0,1,0
    while B==0:
        C=randint ( 1,6 )
        if C<=R:
            A[J]+=1
            B=1
        else:
            J+=1
            R+=1
for J in range ( 6 ) :
    print ( "Fréquence de mort à la {} e place : {} %".format ( J+1, A[J]/N*100 ) )
```

```
Combien d'essais? 100
Fréquence de mort à la 1e place : 15.0%
Fréquence de mort à la 2e place : 28.0%
Fréquence de mort à la 3e place : 24.0%
Fréquence de mort à la 4e place : 19.0%
Fréquence de mort à la 5e place : 13.0%
Fréquence de mort à la 6e place : 1.0%
```

```
Combien d'essais? 10000
Fréquence de mort à la 1e place : 17.05%
Fréquence de mort à la 2e place : 27.49%
Fréquence de mort à la 3e place : 28.02%
Fréquence de mort à la 4e place : 18.39%
Fréquence de mort à la 5e place : 7.5%
Fréquence de mort à la 6e place : 1.55%
```

Essayons un autre algorithme, celui qui a été vu et corrigé en cours :

Lire N — *N est le nombre de fois que l'on veut simuler cette expérience aléatoire*
Initialiser les valeurs d'un tableau contenant 6 nombres (Pour J allant de 1 à 6 : *A*[J]=0)
Pour I allant de 1 à N
    K=1 — *K est le nombre de balles dans le Revolver*
    X=6 — *X est le numéro du barillet choisi pour le tir (les balles sont supposées dans les 1^ers^ numéros)*
    Tant que X > K — *On entre bien initialement dans la boucle et on en sort quand quelqu'un meurt (X≤K)*
        X=nombre entier aléatoire entre 1 et 6 — *On simule le choix aléatoire du numéro du barillet*
        Si X > K Alors {K=K+1} — *Cette dernière instruction (*K=K+1*) ajoute une balle dans le Revolver*
    A[K]=*A*[K]+1
Pour J allant de 1 à 6 : afficher *A*[J]/N

Programmons ce nouvel algorithme, légèrement différent, et plus simple que le premier. Donne t-il les résultats escomptés ? Je le programme en Python car c'est plus facile à transcrire (un copié/collé) et aussi parce que la syntaxe des listes n'est pas bien maîtrisée (de ma part). J'obtiens un résultat qui n'est pas conforme aux attentes : il y a une différence substantielle entre la 2ème et la 3ème place alors que le calcul (et l'algorithme précédent) nous donne des valeurs égales. Qu'est-ce que cela signifie ? Sans doute que notre algorithme contient une erreur dans sa logique, ou dans sa transcription en programme.

```
from random import randint
N=int ( input ( "Combien d'essais? " ) )
A=[0]*7  # pour initialiser un tableau à 7 positions  (la 1ère est numérotée 0)
for I in range ( N)  :
   K,X=1,6
   while X>K :
      X=randint ( 1,6)
      if X>K :  K+=1
   A[K]+=1
for K in range ( 1,7)  :
   print ( "Fréquence de mort à la  {}  e place :   {}   %".format ( K, A[K]/N*100)  )
```

```
Combien d'essais? 10000
Fréquence de mort à la 1e place :  16.81%
Fréquence de mort à la 2e place :  39.14%
Fréquence de mort à la 3e place :  27.05%
Fréquence de mort à la 4e place :  13.13%
Fréquence de mort à la 5e place :  3.53%
Fréquence de mort à la 6e place :  0.34%
```

Il faut bien faire attention pour découvrir, dans ce genre de situation, où se situe l'erreur. Il faut examiner le fonctionnement pas à pas. J'ai fait ce travail et je m'aperçois que quand on augmente K dans l'instruction « if X>K : K+=1 » le test que l'on refait pour voir si on continue la boucle « while X>K » ne donne pas forcément la même bonne réponse. Par exemple, on a K=1 et X=2 donc le « if X>K : K+=1 » incrémente K de 1 (le 1er n'étant pas mort on ajoute une balle) ; on a alors K=2 et X=2 donc « while X>K » n'est plus vrai et on n'entre pas à nouveau dans la boucle, on incrémente alors A[2] alors que ce n'est pas vrai, le 2ème tireur n'a pas encore tiré... Il faut modifier cet algorithme, par exemple en introduisant une variable B, comme précédemment, qui signifie B=1 on s'arrête car quelqu'un est mort (on ne sait pas son rang) et B=0 on continue. On pourrait utiliser une variable booléenne qui a la même fonction (B=True ou B=False). Le programme corrigé donne les résultats attendus, l'erreur était bien là.

```
from random import randint
N=int ( input ( "Combien d'essais? " ) )
A=[0]*7  # pour initialiser un tableau à 7 positions  (la 1ère est numérotée 0)
for I in range ( N)  :
   K,X,B=1,6,0
   while B==0 :
      X=randint ( 1,6)
      if X>K :  K+=1
      else : B=1
   A[K]+=1
for K in range ( 1,7)  :
   print ( "Fréquence de mort à la  {}  e place :   {}   %".format ( K, A[K]/N*100)  )
```

```
Combien d'essais? 10000
Fréquence de mort à la 1e place :  16.39%
Fréquence de mort à la 2e place :  27.93%
Fréquence de mort à la 3e place :  27.65%
Fréquence de mort à la 4e place :  18.15%
Fréquence de mort à la 5e place :  8.37%
Fréquence de mort à la 6e place :  1.51%
```

Je vous laisse découvrir la programmation des listes sur votre calculatrice. Vous pouvez lire ce qui suit pour vous aider ou vous en remettre au manuel de votre calculatrice (c'est plus sûr et cela vous servira par la suite de pouvoir vous y référer).

Un autre bon exercice d'algorithmique :
Simuler le scénario S_2 et déterminer la fréquence des morts en dehors du 1er tour...
Comparer alors avec les données réelles des probabilités (calculées au tableur).

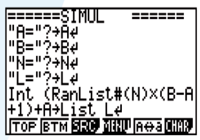
Pour programmer l'utilisation d'un tableau (liste) sur la Casio Graph25+, j'ai trouvé ce qui suit dans le manuel (exercice 2).
NB : je ne sais pas s'il faut faire quelque chose de spécial pour initialiser la liste (je suppose que oui).
Les valeurs sont stockées dans un tableau (une « liste »), c'est ce que l'on voulait, mais comment y accède t-on?
En affichant L[1], L[2], etc., a t-on les valeurs attendues à la fin ? Est-ce plutôt L(1), L(2), etc.? Il faut essayer... ou continuer à chercher sur internet... ou trouver quelqu'un qui sache utiliser cette fonctionnalité bien pratique dans certains cas (mais pas au DS de lundi).

Exercice 2 : Réaliser un programme qui permet de simuler N lancés d'un dé à F faces (F>1) numérotés de A à B (F = B – A + 1).
Les résultats des lancés seront placés dans une liste dont le numéro est L ($1 \leq L \leq 26$)

Pour réaliser le programme nous devons utiliser 4 variables, N pour désigner le nombre de lancer, A pour la valeur minimale des faces du dé, et B la valeur maximale, ainsi que L pour le numéro de la liste.
Pour la génération aléatoire de nombres on utilisera la formule : Int(Ran x (B-A+1)) + A

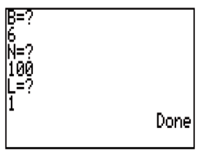
Nous allons créer le programme SIMUL. Dans l'exercice 1 nous avons vu comment demander à l'opérateur de saisir des données.

Nous supposons avoir écrit les 4 premières lignes de programme pour la saisie de A, B, N et L.



Ensuite saisir : OPTN F6 F3 (NUM) F2 (Int) (
OPTN F6 F2 (PROB) F4 (RAND) F5 (LIST)
ALPHA 8 (N)) x (ALPHA log (B) -
ALPHA X,θ,T (A) + 1)) + ALPHA X,θ,T (A)
→ OPTN F1 (LIST) F1 (List ALPHA → (L) EXE

Faire 3 fois EXIT puis exécuter le programme SIMUL et saisir par exemple A = 1 , B = 6, N=100 et L =1 pour avoir dans List 1 un échantillon de 100 nombres compris entre 1 et 6.



Pour les utilisateurs de la TI 82, j'ai trouvé quelque chose d'équivalent, bien que mieux adapté, sur *xmaths.free.fr* (l'auteur marque l'adresse de son site sur toutes les pages de ses documents, donc je me dois

de le nommer ici aussi car il y tient).
Il semblerait donc que ce soit à peu prés le fonctionnement attendu que l'on retrouve sur Algobox.
La liste s'appelle L_1 c'est la 1ère liste (la lettre L, je suppose, est imposé, mais on peut utiliser plusieurs listes en changeant le numéro). Les zones de mémoire sont alors indiquées par un indice : $L_1(1)$ est donc une zone de la mémoire ; $L_1(2)$ en est une autre qui appartient à la même liste. On peut accéder à l'une des deux en écrivant $L_1(A)$, il suffit d'avoir donné la valeur 1 ou 2 à A avant.
Pour incrémenter le contenu de la mémoire $L_1(A)$: $L_1(A)+1 \rightarrow L_1(A)$
Vous constatez que la liste L_1 a été initialisée au début du programme par l'instruction suivante qui code « pour I allant de 1 à 6 en augmentant de 1 à chaque tour de boucle » : For (I,1,6,1) $0 \rightarrow L_1(I)$ End
Pour l'affichage, ce n'est pas beaucoup plus difficile : For (I,1,6,1) Disp $L_1(I)$ End

Un programme pour simuler des lancers de dé

On peut, par un programme, simuler un certain nombre de lancers d'un dé cubique dont les faces sont numérotées de 1 à 6 et donner les résultats.
Le principe est d'appeler autant de fois qu'il faut la fonction int(rand*6)+1 et de noter les effectifs de chacune des faces dans la liste L1 . (Il y a, bien entendu, d'autres façons de faire un tel programme).

Écrire les lignes de code du tableau ci-dessous dans un programme qu'on appellera par exemple DE (la validation et le retour à la ligne se fait avec EXE).
For ; End ; While se trouvent dans [PRGM] CTL (Contrôle)
Input ; Disp se trouvent dans [PRGM] I/O (Entrées-sorties)
> se trouve dans TEST obtenu par [2nd] [MATH]
L1 est obtenu avec [2nd] [1]

http://xmaths.free.fr/

Code	*Explications et Remarques*
`For (I , 1 , 6 , 1)`	Début d'une boucle For pour mettre 0 dans les premières valeurs de L1. La première valeur de I est 1. (I va varier de 1 à 6 par pas de 1)
`0 → L1(I)`	Met 0 dans L1(I). Par exemple si I est égal à 2, L1(I) représente la deuxième valeur de la liste 1, c'est cette deuxième valeur qui contiendra le nombre de faces "2" du dé qui seront tirées.
`End`	Fin de la boucle For. Passage à la valeur de I suivante. Tant que I n'aura pas atteint 6, le programme reprend à la ligne For..... Lorsque I a dépassé 6, le programme passe à la suite.
`Input "NBRE LANCERS ", N`	Entrée du nombre de lancers à simuler.
`While N > 0`	Début d'une boucle pour compter les lancers.
`int(rand*6)+1 → A`	Appel d'une valeur aléatoire, transformation en un nombre entier entre 1 et 6 et stockage dans la mémoire A.
`L1(A)+1→ L1(A)`	Suivant la valeur dans A, on augmente de 1 l'une des six premières valeurs de L1. Par exemple si A est égal à 4 L1(A) représente la quatrième valeur de la liste 1, et c'est cette quatrième valeur qui augmente de 1. (Le nombre de faces "4" augmente de 1)
`N – 1 → N`	Diminution d'une unité du nombre de lancers restant à faire.
`End`	Fin de la boucle While. Tant que N est strictement positif le programme revient à la ligne While. Lorsque N vaut 0, le programme passe à la suite.
`For (I , 1 , 6 , 1)`	Début d'une boucle d'affichage des résultats. La première valeur de I est 1. (I va varier de 1 à 6 par pas de 1)
`Disp L1(I)`	Affichage d'une valeur contenue dans L1 Par exemple si I est égal à 2, L1(I) représente la deuxième valeur de la liste 1, c'est celle qui contient le nombre de faces "2" tirées.
`End`	Fin de la boucle For. Passage à la valeur de I suivante. Tant que I n'aura pas atteint 6, le programme reprend à la ligne For..... Lorsque I a dépassé 6, le programme passe à la suite, c'est-à-dire qu'il est terminé.

```
PROGRAM:DE
:For(I,1,6,1)
:0→L1(I)
:End
:Input "NBRE LAN
CERS",N
:While N>0
:int (rand*6)+1→
```

```
PROGRAM:DE
:int (rand*6)+1→
A
:L1(A)+1→L1(A)
:N-1→N
:End
:For(I,1,6,1)
:Disp L1(I)
```

```
PROGRAM:DE
:End
:For(I,1,6,1)
:Disp L1(I)
:End
:
:
:
```

Lorsque le programme est entré, utiliser QUIT obtenu par [2nd] [MODE] pour revenir à l'écran de calcul.
Pour exécuter le programme utiliser [PRGM] EXEC.
(Contentez vous d'un faible nombre de lancers pour une première utilisation)

```
prgmDE
NBRE LANCERS100
```

```
                18
                17
                25
                 9
                18
                13
              Done
```

Les nombres obtenus correspondent aux effectifs des différentes faces du dé.