Révisions de Python

Dans tout ce TP on supposera qu'on a importé les packages `numpy`, `numpy.random` et `matplotlib.pyplot` par

```
import numpy as np
import numpy.random as rd
import matplotlib.pyplot as plt
```

1 Les bases

Exercice 1. Construire la liste (ou le `np.array`) des $\sqrt{2k}$ pour $k \in [\![0,20]\!]$ de trois façons :

- Avec une boucle `for` en partant d'une liste vide.
- Avec une définition en compréhension.
- Avec la fonction `np.sqrt` et le `np.array [0,1,...,20]`

Exercice 2. Programmer la fonction $f : x \mapsto \begin{cases} 2x^2 \text{ si } x < 0 \\ 3x+1 \text{ si } x \geqslant 0 \end{cases}$.
Tracer sa courbe représentative sur l'intervalle $[-3,3]$.

Exercice 3. Programmer une fonction qui prend en argument un réel x, et renvoie le plus petit entier naturel n tel que $\left|\frac{x}{2^n}\right| < 1$.
(on utilisera une boucle `while`).

Bonus mathématique : pour $x > 0$, exprimer cet entier n en fonction de $\ln(x)$ et $\ln(2)$.

Exercice 4. Soit

```
L = rd.random(10000)
```

Donner une suite de commandes permettant d'afficher un histogramme représentant les éléments de L, en découpant en 20 classes de même largeur, et dont la surface totale des barres vaut 1. Que s'attend-on à observer ?

Exercice 5. Écrire une fonction `somme_des_carres(n)` qui calcule, pour un entier n donnée, la somme $\sum_{k=1}^{n} k^2$
(sans utiliser la formule $\frac{n(n+1)(2n+1)}{6}$.)
On programmera de deux façons :

- À l'aide d'une boucle `for` ;
- En générant la liste des k^2 et en renvoyant la somme des éléments de cette liste.

2 Listes

Exercice 6 (Recherche dans une liste)**.** Programmer une fonction qui prend en arguments une liste L et un réel a et renvoie `True` si a est un élément de L, et `False` sinon.

Exercice 7. Écrire un programme Python qui prend en argument deux listes `L` et `M` et qui renvoie le nombre d'éléments de `L` qui appartiennent à `M`.

Exercice 8. Programmer une fonction `compter`, qui associe à une liste `L` et un réel `a` le nombre d'éléments de `L` égaux à `a`.

Si `L` est un `np.array`, expliquer pourquoi la commande `np.sum(L==a)` renvoie également le nombre d'éléments de `L` égaux à `a`.

Exercice 9 (Valeurs distinctes dans une liste)**.** Programmer une fonction qui prend en argument une liste L, et renvoie une liste qui contient les valeurs distinctes de L, rangées dans leur ordre d'apparition. Par exemple, l'image de `[1,2,1,3,1,-6,1,3]` doit être `[1,2,3,-6]`.
Pour cela on initialisera une liste vide `L1`, et on y ajoutera successivement les valeurs de `L`, sauf lorsqu'une valeur a déjà été rencontrée.

Exercice 10. Programmer une fonction qui prend une liste `L1` et renvoie son « miroir » `L2` : le premier élément de `L1` est le dernier de `L2`, le second de `L1` est l'avant-dernier de `L2`, etc.
Par exemple le miroir de `[1,2,1,3,4]` est `[4,3,1,2,1]`.

On pourra, dans le corps de la fonction, initialiser une liste vide `[]` qu'on remplira par des `.append()` successifs ; on rappelle que `len(L)` désigne le nombre d'éléments d'une liste `L`.

Exercice 11 (Une liste est-elle symétrique ?)**.**
On dit qu'une liste est symétrique si elle est son propre miroir (au sens de l'exercice 10).
Par exemple `[0,4,3,2,3,4,0]` est symétrique ; et `[1,2,3,2,0]` ne l'est pas.

1. Coder une fonction Python appelant la fonction de l'exercice 10, qui renvoie `True` si une liste est symétrique, et `False` si elle ne l'est pas.
2. Coder une fonction Python *n'utilisant pas* la fonction de l'exercice 10, qui renvoie `True` si une liste est symétrique, et `False` si elle ne l'est pas.
   Pour cela, il faut regarder si `L[0]=L[len(L)-1]` ; `L[1]=L[len(L)-2]`,...
   renvoyer `False` dès que l'une de ces égalités n'est pas vérifiée, et `True` si elles le sont toutes.

3 Listes (plus compliqué)

Exercice 12. Recherche dichotomique dans une liste triée.
On dispose d'une liste *triée dans l'ordre croissant* `L` et d'un nombre x ; on veut savoir si x appartient à `L`. On cherche alors à programmer une fonction `recherche_dicho(L,a)` qui renvoie `True` si x est un élément de L et `False` sinon.
La recherche naïve a été vue dans l'exercice 6 ; si la liste est triée on peut aller plus vite par dichotomie.
On regarde l'élément du « milieu » de la liste : c'est l'élément d'indice `int(np.floor(len(L)/2))`

- Si `L[milieu]` vaut x, x est dans L ! On renvoie alors `True`.
- Si `L[milieu]` est $> x$, x ne peut être qu'avant : il faut chercher dans la liste `L[:milieu]`
- Si `L[milieu]` est $< x$, x ne peut être qu'après : il faut chercher dans la liste `L[milieu+1:]`

À chaque tour on garde donc la moitié gauche ou la moitié droite de la liste. Ceci s'arrête si on trouve la composante égale à x, ou si on a épuisé toute la liste...

On délimite par deux indices a et b la tranche de liste à étudier. Le « milieu » de la tranche aura donc pour indice `int(np.floor((a+b)/2))`.
À chaque tour de boucle on met à jour a (si on garde la moitié droite) ou b (si on garde la partie gauche).

Compléter le code suivant :

```
def recherche_dicho(L,x):
    a=0
    b=len(L)-1
    while .... : # tant que la tranche à étudier n'est pas vide
        milieu=int(np.floor((a+b)/2))
        if .... :
            return True
        elif L[milieu]>x:
            ....
        else:
            ....
    return False
```

Exercice 13 (Tri par insertion)**.**
On rappelle les commandes suivantes.

- *Slicing.*
  Si `L` est une liste, la commande `L[a:b]` renvoie la « tranche » de `L` commençant à l'indice `a` et terminant à l'indice `b-1`. Par exemple :

```
>>> L = [1,2,-1,0,-1,5]

>>> L[1:4]
[2,-1,0]
```

- *Concaténation.*
  Si `L1` et `L2` sont deux listes, la commande `L1+L2` renvoie la liste obtenue en mettant `L1` et `L2` bout à bout[1].
  En particulier, la liste `L+[a]` est celle obtenue en ajoutant l'élément `a` à la fin de `L` (peut aussi s'obtenir par `L.append(a)`).

1. Coder une fonction `insertion(i,a,L)` qui insère dans la liste `L`, à la `i`-ème position, la valeur `a`.
2. On suppose que la liste L est triée par ordre croissant. Coder une fonction `insertion_croissante(a,L)` qui insère dans la liste `L` la valeur `a`, en faisant en sorte que la liste obtenue soit encore triée par ordre croissant.
   On pourra utiliser la fonction de la question précédente.
3. On considère une liste `L` qu'on veut trier par ordre croissant. On procède de la manière suivante :
   - On initialise une liste `M`, qui contient pour seul élément le premier élément de `L`.
   - Pour `i` allant de 1 à `len(L)-1`, on insère l'élément d'indice `i` de `L` dans la liste `M`, en faisant en sorte que `M` reste triée par ordre croissant.
   - À la fin de la procédure, `M` est la liste obtenue en triant `L` suivant l'ordre croissant.

   Programmer cette fonction.

[1] Attention, ça ne fonctionnera pas pour des objets de type `np.array` !!

4 Expériences aléatoires

Exercice 14. Programmer de la manière la plus concise possible un script qui renvoie la moyenne de 1000 tirages d'une variable $X \hookrightarrow \mathscr{B}(10, 0.4)$.

Exercice 15. Programmer une fonction (sans argument) qui renvoie un tirage de $Y = e^X$ où $X \hookrightarrow \mathscr{G}(1/3)$.

Exercice 16. Programmer une fonction qui effectue des tirages successifs indépendants d'une variable X suivant la loi de Poisson $\mathscr{P}(4)$, et renvoie la valeur du premier tirage supérieur ou égal à 10.
Modifier cette fonction pour qu'elle renvoie le couple `(a,n)` où `a` est la première valeur supérieure à 10, et `n` le nombre de tirages nécessaires à l'obtention de cette valeur.)

Exercice 17. Programmer un script qui effectue 100 tirages indépendants d'une variable X suivant la loi $\mathscr{G}(1/2)$, et affiche la plus grande valeur obtenue.

Exercice 18. Programmer un script qui simule 100 lancers d'une pièce équilibrée et renvoie le nombre de Pile obtenus. On utilisera seulement la fonction `rd.random`.

5 Expériences aléatoires (plus compliqué)

Exercice 19. Programmer un script qui simule des lancers d'une pièce équilibrée et renvoie le nombre de lancers nécessaires pour obtenir pour la première fois 2 Pile successifs.
On pourra, par exemple, maintenir une variable `i` initialisée à 0, qui est incrémentée de 1 si on obtient Pile, et remise à 0 si on obtient Face.

Exercice 20. On dispose de 10 urnes : pour tout $k \in [\![1, 10]\!]$, l'urne k contient k boules numérotées $1, 2, \ldots, k$.
Écrire un script Python qui simule le choix aléatoire équiprobable d'une urne, puis le tirage d'une boule dans cette urne, et affiche le numéro de la boule tirée.
On utilisera `rd.randint`.

Exercice 21. On considère l'expérience aléatoire suivante (ECRICOME S, 2017) :

On dispose de n urnes initialement vides, numérotées de 1 à n et on dispose d'un grand stock de boules que l'on dépose une à une dans ces urnes. Pour chaque boule, on choisit au hasard, de façon équiprobable, l'urne dans laquelle la boule est déposée. On note X_n le rang du premier tirage pour lequel une des urnes contiendra deux boules.

On modélise cela de la manière suivante :

- On crée une liste L à n composantes, initialement toutes égales à 0 ;
- Chaque tour de boucle correspond à la mise d'une boule dans un urne tirée au sort. On choisit donc aléatoirement cette urne ; et on modifie la liste L pour rendre compte du fait qu'on met la boule choisie dans cette urne.
- Ce processus se répète jusqu'à ce que... ?

Compléter la fonction suivante qui prend en argument le nombre n d'urnes et renvoie un tirage de X_n.

```
def tirage(n):
    L = np.zeros( ... ) # contient le nombre de boules dans chaque urne
    c = 1 # comptera les tirages
    choix_urne = rd.randint( ... , ... ) # choix de l'urne pour la première boule
    while ... :
        L[...] = L[...]+1 # l'urne choisie contient une boule de plus !
        choix_urne = .... #  on prend la boule suivante
        c = .... # et on met à jour le compteur
    return ....
```

Exercice 22. On considère un tournoi réunissant une infinité de joueurs A_0, A_1, A_2,..., A_n,... qui s'affrontent dans une série de duels de la façon suivante :

- A_0 et A_1 s'affrontent durant le duel numéro 1. Le perdant est éliminé du tournoi, le gagnant reste en jeu ;
- le gagnant du premier duel participe au duel numéro 2 durant lequel il affronte le joueur A_2. Le gagnant de ce duel participe au duel numéro 3 contre le joueur A_3 et ainsi de suite ;
- pour tout $k \in \mathbb{N}^*$, le joueur A_k participe au duel numéro k, qu'il peut remporter avec une probabilité $p \in]0,1[$; son adversaire durant ce duel pouvant également remporter le duel avec la probabilité $q = 1-p$.
- est désigné gagnant du tournoi, le premier joueur, s'il y en a un, qui gagne N jeux successifs lors du tournoi.

Compléter la fonction suivante pour qu'elle simule ce tournoi et renvoie le numéro du joueur gagnant (on admet qu'il y a presque sûrement un gagnant).

```
def tournoi(N,p):
    j = 0 # joueur gagnant
    c = ... # son compteur de victoires
    k = 1 # compteur de duels
    while ... :
        if ... : # le joueur k gagne le match k
            j = ...
            c = ...
        else:
            c = ...
            k = ...
    return ...
```

Solutions

1

```
L1=[]
for k in range(21):
    L1.append(np.sqrt(2*k))

L2= [np.sqrt(2*k) for k in range(21)]

L3=np.sqrt(2*np.arange(21))
```

2

```
def f(x):
    if x<0:
        return 2*x**2
    return 3*x+1

X=np.linspace(-2,2,1000)
Y=[f(x) for x in X]
plt.plot(X,Y)
plt.show()
```

3

```
def f(x):
    n=0
    while abs(x/2**n)>=1:
        n=n+1
    return n
```

4

```
L=rd.random(1000)
plt.hist(L,20,density=True)
plt.show()
```

5

```
def somme_des_carres(n):
    S=0
    for k in range(1,n+1):
        S=S+k**2
    return S
```

ou

```
def somme_des_carres(n):
    return sum(np.arange(1,n+1)**2)
```

6

```
def appartient(L,a):
    for l in L:
        if l==a:
            return True
    return False
```

7

```
def fonc(L,M):
    c=0
    for k in L:
        if k in M:
            c=c+1
    return c
```

8

```
def count(L,a):
    c=0
    for l in L:
        if l==a:
            c=c+1
    return c
```

9

```
def supprime_doublon(L):
    L1=[]
    for l in L:
        if l not in L1:
            L1.append(l)
    return L1
```

10

```
def miroir(L):
    n=len(L)
    M=[]
    for k in range(n):
        M.append(L[n-1-k])
    return(M)
```

11

1.
```
def symetrique1(L):
    if L == miroir(L):
        return True
    return False
```

ou (pour les esthètes) :

```
def symetrique1(L):
    return L == miroir(L)
```

2.
```
def symetrique2(L):
    for k in range(int(len(L)/2)):
        if L[k]!=L[len(L)-1-k]:
            return False
    return True
```

12

```
import numpy as np
def recherche_dicho(L,x):
    a=0
    b=len(L)-1
```

```
    while b-a>1:
        milieu=int(np.floor((a+b)/2))
        if L[milieu]==x:
            return True
        elif L[milieu]>x:
            b=milieu
        else:
            a=milieu+1
    return False
```

13

1.
```
def insertion(i,a,L):
    return L[:i]+[a]+L[i:]
```

2. Ici il faut trouver la place à laquelle insérer notre valeur : c'est l'indice du premier élément de la liste supérieur ou égal à `a` (ou à la fin, si tous les éléments sont `<a`).
On obtient cette position par une boucle `while`, en prenant garde à ne pas sortir de la liste avec des indices trop grands !
(voir : évaluation paresseuse des conditions).
```
def insertion_croissante(a,L):
    i=0
    while i<len(L) and L[i]<a:
        i=i+1
    return insertion(i,a,L)
```

3.
```
def tri_insertion(L):
    M=[L[0]]
    for k in range (1,len(L)):
        M=insertion_croissante(L[k],M)
    return M
```

14
```
np.mean(rd.binomial(10,0.4,1000))
```

15
```
def exp_geometrique():
    return np.exp(rd.geometric(1/3))
```

16
```
def poisson_sup_10():
    a=rd.poisson(4)
    while a<10:
        a=rd.poisson(4)
    return a
```

Avec la variante
```
def poisson_sup_10():
    a=rd.poisson(4)
    n=1
    while a<10:
        a=rd.poisson(4)
        n=n+1
    return (a,n)
```

17

```
print(max(rd.geometric(0.5,100)))
```

18

```
n=0
for k in range(100):
    if rd.random()<1/2:
        n=n+1
print(n)
```

19

```
def deux_pile():
    n=0
    i=0
    while i<2:
        if rd.random()<1/2:
            i=i+1
        else:
            i=0
        n=n+1
    return n
```

20

```
urne=rd.randint(1,11)
boule=rd.randint(1,urne+1)
```

21

```
def tirage(n):
    L = np.zeros(n)
    c = 1
    choix_urne = rd.randint(1,n+1)
    while L[choix_urne-1] == 0 : # l'urne dans laquelle on souhaite mettre la
                                 # boule est vide
        L[choix_urne-1] = L[choix_urne-1]+1
        choix_urne = rd.randint(1,n+1)
        c = c+1
    return c
```

(Rappel sur l'origine des indices : l'urne numéro 1 correspond à la première composant de L, donc à `L[0]`. il faut donc incrémenter `L[choix_urne-1]`).

22

```
def tournoi(N,p):
    j = 0 # joueur gagnant
    c = ... # son compteur de victoires
    k = 1 # compteur de duels
    while c<N:
        if rd.random()<p: # le joueur k gagne le match k
            j = k
            c = 1
        else:
            c = c+1
            k = k+1
    return j
```