

Activité 2

Indispensables listes...

1 Quelques rappels « théoriques »...

1.1 Listes python

Sous Python, on peut définir une liste comme une collection ordonnée d'éléments séparés par des virgules, l'ensemble étant enfermé dans des crochets. Les différents éléments ne sont pas forcément de même type.

Chaque élément est repéré par son indice (ou index, ou rang), mais attention, le premier élément est celui d'indice 0, le deuxième d'indice 1 etc...

ATTENTION - ATTENTION - ATTENTION - ATTENTION

L'indice commence à 0!

ATTENTION - ATTENTION - ATTENTION - ATTENTION

c'est malain.

Fonctions de base de manipulation de listes :

<code>maliste = []</code>	crée une liste vide nommée « maliste »
<code>maliste.append(9)</code>	ajoute l'élément « 9 » en queue de liste
<code>len(maliste)</code>	renvoie le nombre d'éléments (la longueur) de la liste
<code>maliste[i]</code>	renvoie l'élément de rang i contenue dans maliste

Un peu plus tard, on compliquera en introduisant les listes de listes.

1.2 Accès aux éléments d'une liste

a Accès à l'aide d'un indice

La solution la plus simple pour obtenir un indice permettant d'accéder à tous les éléments d'une liste L est :

```
for i in range(len(L)) :
```

On peut alors accéder à l'élément d'indice i via $valeur = L[i]$ ou le modifier via $L[i] = valeur$

Si on ne souhaite pas accéder à tous les éléments de la liste, on utilisera une boucle *for* « classique » :

```
for i in range(début, fin (valeur exclue), pas (= 1 par défaut)) :
```



ATTENTION : les variables *début*, *fin* et *pas* sont obligatoirement des entiers.

ATTENTION : la valeur *fin* n'est pas atteinte :

Vous verrez chez certains de vos élèves l'accès à un élément d'une liste avec des indices négatifs. Si $L = [1,2,3]$, $L[-1]$ vaut 3, $L[-2]$ vaut 2. Ce n'est pas, à mon humble avis, une démarche à recommander...

b

Accès sans indice

Le parcourt des éléments d'une liste L (sans accéder aux indices) peut se faire directement par l'instruction `for x in L`. Dans ce cas, la variable x va prendre successivement (à chaque itération) la valeur de chacun des éléments de la liste (qui ne sont pas forcément des nombres).



On accède, ainsi, directement à la variable x (on ne dispose pas de son indice dans le tableau) dont on ne peut pas changer la valeur

c

« Slicing »

Bien sûr, ce n'est pas un scoop, on accède à l'élément d'indice i du tableau T à l'aide de l'instruction $T[i]$ en faisant, attention :

- au fait que les indices commencent à ZÉRO ;
- que le dernier élément du tableau a donc comme indice $\text{len}(T) - 1$;
- qu'un indice hors limite est facilement détecté à l'exécution, mais que par écrit... c'est le correcteur qui aura la joie de repérer vos « index out of range ! ». Oh ! combien de points perdus là-dessus!!!

Parfois on ne souhaite par récupérer un élément unique de la liste mais plutôt une sous-liste de la liste d'origine.

On peut faire ça « à la main » à l'aide de notre bonne vieille instruction `for i in range(debut, fin, pas)` !

```

1  T = [1,2,3,4,5,6,7,8,9]
2  # les 4 premiers éléments du tableau
3  x = [T[i] for i in range(0,4,1)]
4  # tous les éléments à partir du 3ème
5  y = [T[i] for i in range(2,len(T),1)]
6  # les 5 derniers éléments du tableau
7  z = [T[i] for i in range(len(T)-5,len(T),1)]
8  # 3 éléments du tableau à partir du 5ème
9  t = [T[i] for i in range(4,7,1)]
10 # un élément sur 3 à partir du second
11 u = [T[i] for i in range(1,len(T),3)]

```

Bien sûr, on peut omettre le paramètre 1 comme valeur du pas et le 0 comme premier indice dans certaines expressions.

Pour faire plus savant dans les salons de thé, on peut aussi faire du « slicing » ; littéralement, du « découpage en tranche » grâce à l'instruction $T[\text{debut} : \text{fin} : \text{pas}]$. Jusque là, pourquoi pas ; mais il faut faire attention aux raccourcis :

- si le *pas* vaut 1 on peut omettre d'écrire : 1 à la fin ;
- si le début vaut 0, on peut omettre d'écrire sa valeur (mais il faut quand même garder les « : ») ;

- si la fin est le dernier élément de la liste, on peut omettre d'écrire sa valeur (mais il faut garder les « : » qui précède).

On peut ainsi proposer le même programme que précédemment mais avec « : » au lieu de `in range(...)`.

Listing 2.1 – Extraction des éléments par slicing

```

1  T = [1,2,3,4,5,6,7,8,9]
2  # les 4 premiers éléments du tableau
3  x = T[:4]
4  # tous les éléments à partir du 3ème
5  y = T[2:]
6  # les 5 derniers éléments du tableau
7  z = T[len(T)-5:]
8  # 3 éléments du tableau à partir du 5ème
9  t = T[4:7]
10 # un élément sur 3 à partir du second
11 u = T[1::3]
```

Cette technique de slicing est au programme donc vous verrez, peut-être, quelques uns de vos élèves l'utiliser. En ce qui me concerne, je ne l'utilise pas... mais c'est peut-être mon côté « anti raccourcis de matheux! ».

1.3 Création d'une liste

```

1  # création "à la main" de vecteurs
2  L = [4,3,2,1]
3  T = [8,2,5,4,1,3,7,0]
4  # ou encore pour la même liste L
5  L = []
6  for i in range(4):
7      L.append(4-i)
8  # ou encore
9  L = []
10 for i in range(4,0,-1):
11     L.append(4-i)
12 # ou encore, en plus compact...
13 L = [4-i for i in range(4)]
14 #on peut aussi créer une liste de 0 de la même façon
15 T = [0 for i in range(10)]
16 # et pourquoi pas un tableau de valeurs pour une fonction
17 S = [sin(i*pi/100) for i in range(100)]
```

La siouxerie proposée sur les trois derniers exemples (liste dite par compréhension) consiste à remplir une liste à partir des éléments d'une autre liste en effectuant, éventuellement, une opération.

En cachette, l'instruction `range(debut, fin(exclue), pas)` crée une liste `[debut, debut + pas, debut + 2 pas, ...]`. Et `for i in machin` va successivement chercher tous les éléments de la liste `machin` pour les stocker dans la variable `i`. Ainsi `i` vaut, successivement : `debut`, `debut + pas`, `debut + 2 pas`, ... et la partie gauche de l'instruction décrit, en fait, quelle fonction (de `i`) utiliser pour remplir la liste : en a en fait : `[f(i) for i in machin]`.



Il faut juste avoir conscience que plusieurs solutions existent... vous utilisez celle que vous voulez, mais pas un patchwork ! Et, si vous avez peur de vous tromper : vous créer une liste vide et vous

faites un `append()` !

1.4 Pas d'opération arithmétique sur les listes !

Il faut absolument se garder de faire toute opération arithmétique sur les listes !

```
1 L1 = [1,2,3] + [4,5,6]
2 L2 = 3*[1,2,3]
3 L3 = sqrt([1,2,3])
```

Les deux instructions précédentes créent deux listes :

- L1 correspond à [1,2,3,4,5,6] ;
- L2 correspond à [1,2,3,1,2,3,1,2,3]
- la troisième ligne conduit à une erreur de syntaxe

L'utilisation des deux premières expressions n'est pas interdite, elle conduit la concaténation de listes.

1.5 Quelques mots sur NumPy

NumPy est une bibliothèque fondamentale dans l'utilisation de python pour des applications scientifiques ; surtout si on y associe sa copine SciPy. La plupart des méthodes numériques utiles au calcul scientifique sont présentes dans ces bibliothèques. On fera une séance particulière sur NumPy plus tard !

2 A vous de jouer...

2.1 Blondes

- .1 A l'aide d'une boucle, remplir une liste :
 - a) qui contient tous les nombres entiers de 0 à 10 inclus ;
 - b) qui contient tous les nombres entiers pairs de 10 à -10 inclus ;
 - c) qui contient tous les nombres compris entre 0 et 10 (inclus) espacés de 0,1.
 - d) qui contient les 26 lettres minuscules de l'alphabet.

- .2 Initialiser les 3 tableaux suivants :

```
1 Eleves = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J"]
2 Trim1 = [10, 12, 9, 14, 8, 15, 18, 13, 6, 10]
3 Trim2 = [11, 12, 8, 16, 12, 16, 16, 11, 8, 9]
```

Ecrire le code d'un programme permettant de :

- a) déterminer le nombre d'élèves ayant progressé du premier au second trimestre ;
- b) calculer la moyenne du premier trimestre ;
- c) déterminer la meilleure note du premier trimestre ;
- d) déterminer l'élève qui a le meilleur note au premier trimestre ;
- e) déterminer la listes des élèves qui ont la meilleure note au second trimestre.

2.2 Châtain clair...

- .1 **Nombres premiers** Construire la table des nombres premiers inférieurs à $N = 1000$.

- première méthode : on construit la table, nombre premier par nombre premier en testant chaque nouveau nombre susceptible d'être premier.
- deuxième méthode : on utilise la méthode du crible d'ERATOSTHÈNE (on élimine d'une table des entiers de 2 à N tous les multiples d'un entier premier).

.2 Tri Triez, par une méthode de votre choix (autre que `sort`!), le tableau *Trim1* défini précédemment par valeurs croissantes. Il faudrait si possible trier « sur place » sans créer un nouveau tableau.

2.3 Réactions successives (Acte I) !

On considère le célèbre système $A \longrightarrow B \longrightarrow C$ ¹.

On propose une approche statistique en modélisant le système par un **automate cellulaire**. Partant d'une population de N entités, on va voir, pour chacune d'elle quelle est sa probabilité de disparaître (ou d'apparaître). On note p la probabilité de disparaître par unité de temps $\Delta t = 1$ (unité arbitraire).

Pour chaque entité du système, considérée comme un réactif, on tire un nombre au hasard (entre 0 et 1 grâce à l'instruction `rd.random()`), si ce nombre est supérieur à p l'entité se transforme, sinon il ne se passe rien².

On réitère alors le processus un certain nombre de fois.

On définit, ici, deux probabilités de transitions p_1 (de A vers B) et p_2 de (B vers C).

Afin de tracer l'évolution des quantités en fonction du temps, on stocke (tous les Δt) dans 4 tableaux : T, A, B, C respectivement le temps, le nombre de molécules A, B et C.

Voici le début, et la fin du programme à réaliser... yapluka !

```

1  import random as rd
2  import matplotlib.pyplot as plt
3
4  #paramètres du système
5  N = 10000
6  p1 = 0.90
7  p2 = 0.95
8  tmax = 100
9
10  """ A vous de jouer """
11
12
13  plt.plot(T, A, 'r')
14  plt.plot(T, B, 'g')
15  plt.plot(T, C, 'b')
16  plt.show()
```

1. Aïe, Corinne, pas sur la tête! Bon, OK, je contextualise... en DS je posais une filiation radioactive utilisée en médecine nucléaire où le $^{99}_{42}\text{Mo}$ (noté A) est précurseur du $^{99}_{43}\text{Tc}$ (noté C). Un intermédiaire métastable (noté B) est formé par une première transformation de demi-vie 66 h et décomposé par une seconde transformation de demi-vie 6 h.

2. Pour des réactions d'ordre 1 : $p = 1 - e^{-k\Delta t}$ avec $\Delta t = 1$ (unité arbitraire) correspondant à 1 tirage au sort

3 Quelques pistes...

3.1 Blondes

.1 A tout hasard... la table des codes ASCII permet de passer d'une lettre à un nombre et vice versa.

.2

- a) fastoche!
- b) un jeu d'enfant.
- c) idem.
- d) ne me dites pas que vous n'y arrivez pas!
- e) un peu plus malin! Essayez de récupérer la liste des meilleurs élèves à l'aide d'une seule boucle!

3.2 Châtain clair...

.1 Nombres premiers

- On ajoute dans la liste des nombres premiers en cours de construction tout nombre qui n'admet pas de diviseur dans cette liste de nombre premiers. On pourra utiliser l'opérateur modulo (%) : $a \% b$ renvoie le reste de la division euclidienne de a par b . Ainsi : $a \% 2 = 0$ si a est pair, 1 si a est impair.
- L'idée est de partir d'une liste qui contient $[0, 0, 2, 3, 4 \dots, N]$. On parcourt la table en mettant à 0 tous les multiples d'un nombre non nul. Et enfin, on récupère les nombres non nuls!

.2 **Tri** Un des plus simples à programmer est le tri par sélection :

- on prend le premier élément de la liste ; on le permute avec le plus petit élément de la liste (à sa droite) ;
- on passe au second que l'on permute avec le plus petit élément de liste (à sa droite) ;
- et on recommence !

3.3 Réactions successives

Une fois la modélisation du système effectuée, on se retrouve, classiquement devant deux difficultés :

- comment implémenter le modèle proposé ?
- comment résoudre ?

Avec un peu d'habitude, et pour des systèmes pas trop complexes, vous deviendrez des as en programmation et il restera juste à se poser les questions : comment modéliser ? et quelle est l'implémentation la plus simple du modèle ?

Bref, ici, on a manifestement 3 états (A, B et C). Le plus simple est de coder cet état par un entier valant respectivement (0, 1 ou 2). L'automate sera alors implémenté par une liste de N entiers, initialement initialisés à 0.

Tous les $\Delta t = 1$, et pour chaque cellule, on tire un nombre au hasard entre 0 et 1 :

- si la cellule est dans l'état 0 et que le nombre est supérieur à p_1 , elle passe dans l'état 1 ;
- si la cellule est dans l'état 1 et que le nombre est supérieur à p_2 , elle passe dans l'état 2 ;
- si la cellule est dans l'état 2... il n'y a rien à faire

Il reste alors à compter combien on a de 0, de 1 et de 2 et stocker ces valeurs dans chaque tableau.

4

Solutions...

4.1

Blondes

```

.1
1 L1 = []
2 for i in range(0,11):
3     L1.append(i)
4 # ou L1 = [i for i in range(0,11)], idem pour les suivantes
5
6 L2 = []
7 for i in range(10,-12,-2):
8     L2.append(i)
9
10 L3 = []
11 x = 0
12 while x <= 10 :
13     L3.append(x)
14     x = x + 0.1
15
16 # ou
17 #L3 = []
18 #for i in range(0,101):
19 #    L3.append(i/10)
20
21 L4 = []
22 for i in range(ord("a"), ord("a")+26):
23     L4.append(chr(i))
24 print(L4)

```

- au début, on construit les listes avec des `append` puis, avec l'habitude, on utilise les listes par compréhension !
- `ord` récupère le caractère ASCII de la lettre, on incrémente ce code et on récupère la caractère à l'aide de `chr`

```

.2
1 #a)
2 n = 0
3 for i in range(len(Trim1)):
4     if Trim2[i] > Trim1[i]:
5         n = n + 1
6 print ("Nombre de progrès : ", n)
7
8 #b)
9 s = 0
10 for note in Trim1 :
11     s = s + note
12 moyenne = s / len(Trim1)
13 print ("Moyenne : ", moyenne)
14
15 #c)
16 maxi = Trim1[0]

```

```

17 for i in range(1, len(Trim1)):
18     if Trim1[i] > maxi :
19         maxi = Trim1[i]
20 print("Meilleure note : ", maxi)
21
22 #d)
23 maxi = Trim1[0]
24 indice = 0 # ou best = Eleves[0]
25 for i in range(1, len(Trim1)):
26     if Trim1[i] > maxi :
27         maxi = Trim1[i]
28         indice = i # ou best = Eleves[i]
29 print("Meilleure note : ", maxi, " pour ", Eleves[indice])
30 # ou print("Meilleure note : ", maxi, " pour ", best)
31
32 #e)
33 maxi = Trim2[0]
34 best = [Eleves[0]]
35 for i in range(1, len(Trim2)):
36     if Trim2[i] == maxi :
37         best.append(Eleves[i])
38     elif Trim2[i] > maxi :
39         maxi = Trim2[i]
40         best = [Eleves[i]]
41 print("Meilleure note : ", maxi, " pour ", best)

```

- a) rien à dire puisque je ne vous parle plus des problèmes d'indices !
- b) un classique
- c) on initialise maxi, ici, avec la valeur du premier élément de la liste, on aurait tout aussi bien pu prendre -1 comme valeur.
- d) Il faut, bien sûr, repérer en même temps que le maximum, l'indice correspondant (ou directement le nom dans la table Eleves).
- e) Une solution « bourrin » consisterait à rechercher la meilleure note puis de parcourir la liste Eleves à la recherche des élèves qui ont cette note. En une seule boucle, on fait comme précédemment, simplement best n'est plus un nom (ou un indice) mais la liste des noms. Si on trouve une meilleure note, il faut réinitialiser cette liste (ligne 40).

4.2

Châtain clair...

.1 Nombres premiers

```

1 N = 10000
2
3 start = time.time()
4 P = []
5 for i in range (2,N+1):
6     trouve = False
7     j = 0
8     while j < len(P) :
9         if i%P[j] == 0:
10             trouve = True
11             j = j + 1
12     if not trouve :
13         P.append(i)

```



```

14 interval = time.time() - start
15 print (P)
16 print (interval)
17
18 start = time.time()
19 E = [0,0]# 0 et 1 ne sont pas premiers !
20 for i in range(2, N+1):
21     E.append(i)
22 for i in range(2, int(sqrt(N))+1):
23     if E[i] != 0 : # c'est un nombre premier
24         for j in range(2*i, N+1, i):
25             E[j]= 0
26 Pr = []
27 for i in range(N+1):
28     if E[i]!= 0 :
29         Pr.append(i)
30 interval = time.time() - start
31 print (Pr)
32 print (interval)

```

Pour le fun, j'ai ajouté un décompte du temps nécessaire à l'exécution d'un bloc de code. La méthode « bourrin » prend (pour $N = 1000$) 3,41 s alors que celle utilisant le crible d'ERTOSTHÈNE ne prend que 5,4 ms (sur mon ordinateur).

Sinon, initialiser le tableau E avec 0 et 0 (pour les indices 0 et 1) permet d'avoir l'indice d'un nombre égal à ce nombre.

.2 tri

```

1 for i in range(len(Trim1)):
2     mini = Trim1[i]
3     indice = i
4     for j in range(i+1, len(Trim1)):
5         if Trim1[j] < mini :
6             mini = Trim1[j]
7             indice = j
8     tmp = Trim1[i]
9     Trim1[i] = mini
10    Trim1[indice] = tmp
11 print (Trim1)

```

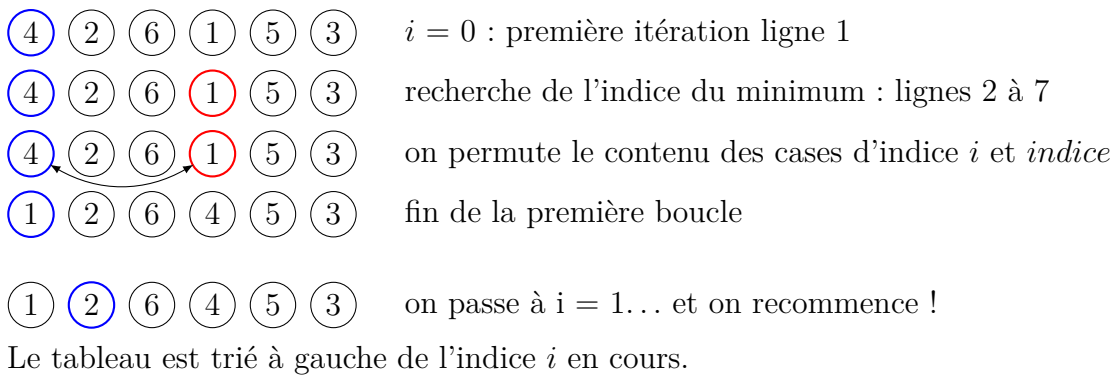
Il faut, bien sûr, deux boucles imbriquées ici :

- la première, indice i , permet de pointer vers la position que l'on est en train de traiter dans la liste ;
- la seconde, indice j , permet de chercher le minimum dans la liste à droite de i .

Lorsque ce minimum est trouvé, on permute le contenu de $L[i]$ et $L[\text{indice du minimum}]$. Une variable intermédiaire tmp est nécessaire pour ne pas perdre la valeur de $Trim1[i]$ ³.

Ci dessous un exemple (avec moins de valeurs pour que ça tienne en largeur) :

3. Vous verrez certains de vos élèves utiliser une astuce python pour permuter 2 nombres ; il suffit simplement d'écrire $a,b = b,a$ et le tour est joué ! Je n'aime pas trop ce type de raccourci car c'est vraiment spécifique à ce langage).



4.3

Réactions successives

```

1  import random as rd
2  import matplotlib.pyplot as plt
3
4  #paramètres du système
5  N = 10000
6  p1 = 0.90
7  p2 = 0.95
8  tmax = 100
9
10 #initialisation de l'automate et des tableaux
11 Automate = [0 for i in range(N)]
12 T = [0]
13 A = [N]
14 B = [0]
15 C = [0]
16
17 t = 0
18 while t < tmax :
19     #on gère les transitions
20     for i in range(N):
21         x = rd.random()#nombre aléatoire entre 0 et 1
22         if Automate[i] == 0 and x > p1 :
23             Automate[i] = 1
24         elif Automate[i] == 1 and x > p2 :
25             Automate[i] = 2
26     #on compte le nombre d'états
27     nbA = 0
28     nbB = 0
29     nbC = 0
30     for i in range(N):
31         if Automate[i]==0 :
32             nbA = nbA+1
33         elif Automate[i]==1 :
34             nbB = nbB+1
35         else :
36             nbC = nbC + 1
37     # on n'oublie pas d'incrémenter le temps
38     t = t+1
39     #on stocke les valeurs 1 fois sur 10
40     T.append(t)
41     A.append(nbA)

```

```
42     B.append(nbB)
43     C.append(nbC)
44
45 plt.plot(T, A, 'r')
46 plt.plot(T, B, 'g')
47 plt.plot(T, C, 'b')
48 plt.show()
```

Ce n'est finalement pas si compliqué...

- on définit les paramètres du système ;
- on initialise les variables nécessaires. Par rapport à ce qui était donné dans l'énoncé, il faut « juste » rajouter l'automate !
- on fait une boucle (ici une boucle `for t in range(1, tmax+1)` aurait été tout à fait possible) ;
- dans cette boucle on teste les transitions pour chacune des cellules
- et on compte combien on en a.
- Il n'y plus qu'à tracer les courbes !