Informatique Pour Tous

TD 6 – Les Listes

Dans cette activité, on s'intéresse à des listes contenant uniquement des nombres (types *int* ou *float*).

A / Calculs simples sur des liste de nombres

Pour tester vos fonctions, vous copie-collerez cette variable dans votre espace des variables globales :

```
L_test = [1, 6, 4, 7, 4, -1, -4, 0, 4, 3]
```

Q1 – Écrire une fonction `occurrence(a, L)` prenant en argument une variable *a* et une liste de nombres *L* quelconque, et renvoyant le nombre d'occurrences de *a* (nombre de fois que *a* se trouve) dans la liste *L*.

Tests : `print(occurrence(4, L_test))` → ceci devrait vous afficher « 3 », car « 4 » y apparaît 3 fois
`print(occurrence(2, L_test))` → ceci devrait vous afficher « 0 », car 2 n'y apparaît pas
`print(occurrence(3, []))` → ceci devrait vous afficher « 0 », car la liste est vide

On admet que les listes pour lesquelles les fonctions suivantes seront appelées sont **non vides**.

Q2 – Écrire une fonction `maxi(L)` prenant en argument une liste de nombres *L* non vide, et renvoyant la valeur du plus grand élément de la liste.

Test : `print( maxi(L_test) )` → ceci devrait vous afficher « 7 ».

Contrainte : je vous impose l'utilisation d'une boucle `for element in L :` `# pas de range()`

Q3 – Écrire une fonction `indice_mini(L)` prenant en argument une liste de nombres *L* non vide, et renvoyant la valeur de **l'index** (= position) du plus petit élément de la liste.

Test : après la définition de cette fonction, dans l'espace d'appel des fonctions et des variables globales, tapez :
`print( indice_mini(L_test) ).`
→ ceci devrait vous renvoyer « 6 ». En effet, le mini est « -4 » et se trouve en 6ème position dans `L_test` (eh oui,Python commence à 0)

Contrainte : je vous impose l'utilisation d'une boucle `for i in range(...) :` `# range() obligé`

On rappelle que si les $(x_n)_{1\leq n\leq N}$ sont les *N* éléments d'une liste, alors la moyenne $\bar{x}$ se calcule par :

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

et l'écart-type s se déduit du calcul de moyenne par la formule :

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Q4 – Écrire une fonction `moyenne(L)` prenant en argument une liste de nombres *L* non vide, et renvoyant la valeur de la moyenne des éléments de cette liste.

Contrainte : vous devez coder la somme vous-même (une boucle *for*), l'usage de `sum(L)` est évidemment proscrit.

Test : `print(moyenne(L_test))` → ceci devrait vous afficher « 2.4 »

Q5 – Écrire une fonction `ecart_type(L)` prenant en argument une liste de nombres *L* non vide, et renvoyant la valeur de l'écart-type des éléments de cette liste.

Aide : en amont de votre code, vous taperez la commande `import math as m` auquel cas, dans la suite, la racine sera disponible avec la commande `m.sqrt(…)`

Test : `print(ecart_type(L_test))` → ceci devrait vous afficher « 3.373096… »

B / Générer des listes de nombres simples

Rappel : pour générer une liste, pour le moment une seule méthode est autorisée :

```
Liste = []    # on déclare la liste vide
while ou for ... :    # puis, de façon itérative
    Liste.append( ... )    # on insère un élément à droite de cette liste
```

Pour tester vos fonctions, vous copie-collerez cette variable dans votre espace des variables globales :

```
L_test2 = list(range(10))    # vaut [0,1,2,3, …, 8,9]   (taille 10)
```

Q6 – Écrire une fonction `retourne(L)` prenant en argument une liste *L* et renvoyant une liste de même taille que *L*, correspondant au renversement de la liste initiale.

Tests : `print(retourne(L_test2))` → ceci devrait vous afficher `[9,8,7,6, …,2,1,0]`
Si vous faites le test sur une liste de taille 1 (**ex** : `[4]`) ou 0 (donc une liste vide `[]`), le renversement devrait renvoyer la liste initiale.

Second exemple, nous voulons générer une liste de nombres premiers. Pour cela nous avons besoin d'une fonction auxiliaire. On utilisera la définition suivante pour un nombre premier. Un nombre $N \in \mathbb{N}$ est premier ssi aucun entier compris entre 2 et $E(\sqrt{N})$ n'est multiple de N.

Q7 – Écrire une fonction `est_premier(N)` prenant en argument un entier *N* et renvoyant un booléen : `True` si *N* est premier, `False` sinon.

Aide : la partie entière d'un flottant *x* (type *float*) s'obtient avec la commande `int(x)`

Contrainte : la fonction **doit** tenir en 5 lignes, en-tête (ligne `def`…) et renvoi compris !

Tests : tester votre fonction avec des entiers simples : 2, 3, 11, 13 sont premiers ; 4, 6, 9, 12 non.

Q8 – Écrire alors une fonction `premiers(M)` prenant en argument un entier *M* et qui renvoie la liste ordonnée (par ordre croissant) de tous les nombres premiers compris entre 2 et M.

Aide : votre algorithme devra initialiser $N = 2$ et s'appuyer sur une boucle `while N <= M :`

Contrainte : la fonction **doit** tenir en 8 lignes, en-tête et renvoi compris.

Test : `print(premiers(40)) # doit afficher [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]`

Comme vous le verrez en physique et en SI, pour la plupart des problèmes que nous rencontrons en sciences, une solution peut être approchée par calcul numérique, au moyen de suites qui convergent vers la solution recherchée. Les suites sont donc extrêmement importantes pour nous.

Soit une suite $(U_n)_{n\geq 0}$. Pour chaque $n \in \mathbb{N}$, l'expression de U_n peut être :

- **Explicite**, c'est-à-dire qu'elle peut se mettre sous la forme $U_n = f(n)$. Exemple : $U_n = \frac{3n+2}{\log(n+2)}$
- **Implicite,** le plus souvent connue *via* une relation de récurrence $U_n = f(U_{n-1})$: Exemple : $U_{n+1} = (2U_n + 1)^2$

Supposons que, par la suite, on souhaite ranger N termes dans une liste `U` les termes $[U_0, U_1, \dots U_{N-1}]$, où *N* est fixée à l'avance. Ainsi, pour tout $k \leq N - 1$, on aura `U[k]` qui correspondra au terme U_k.

B.1 – Suite connue explicitement

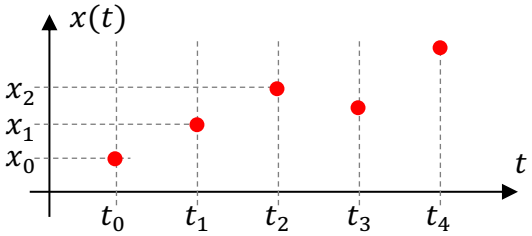
Prenons par exemple une suite arithmético-géométrique. Soient $(a, b) \in \mathbb{R}^2$, et posons $(U_n)_{n\in\mathbb{N}}$ telle que

$$\forall n \in \mathbb{N}, U_n = an + b$$

Q9 – Écrire une fonction `SAG(a,b,M)` prenant en argument deux paramètres a et b, ainsi qu'un rang M, et qui renvoie une liste des valeurs successives de la suite (U_n), jusqu'à U_M.

Test : `print(SAG(2,1,5))` → ceci devrait vous afficher `[1, 3, 5, 7, 9, 11]`

Voyons un exemple : la dérivée. Soit une liste de temps $(t_i)_{0\leq i\leq N}$ (qui compte $N+1$ points), et une liste de positions $(x_i)_{0\leq i\leq N}$, telle que pour tout $i \in [\![0, N]\!]$, $x_i = x(t_i)$ la position relevée à l'instant $t = t_i$.



Supposons qu'on veuille calculer les vitesses $v_i = v(t_i) = \frac{dx}{dt}(t_i)$, et ranger tous les termes dans une liste `V` pour laquelle `V[i]` correspondra à $v(t_i)$.

On admet que si les t_i sont rapprochés d'un pas de temps court, la dérivée peut être approchée par le taux d'accroissement :

$$v_i = \frac{dx}{dt}(t_i) \approx \frac{x_{i+1} - x_i}{t_{i+1} - t_i}$$

Remarque : quelle que soit la valeur de i, les $x_i, x_{i+1}, t_i, t_{i+1}$ sont connus, cette expression de v_i est donc explicite.

Q10 – Écrire une fonction `derivee(LT, LX)` qui prend en argument une liste des temps (t_i) et une liste des positions correspondantes (x_i). On admet que ces deux listes sont de même taille $N+1$. Et la fonction renverra une liste des $(v_i)_{0\leq i\leq N-1}$ vitesses correspondantes.

Remarque : la liste en sortie compte un terme de moins que les listes en entrée, puisque v_N ne peut pas être calculée, en effet x_{N+1} et t_{N+1} n'existent pas.

Test : Soient (à copier coller)
```
Temps = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
Position = [1,4.5,7.5,10,11,11.5,11.5,10,7,4,0]
```
Alors la commande `print(derivee(Temps, Position))`

… aux erreurs d'arrondis près, ceci devrait afficher : `[35, 30, 25, 10, 5, 0, -15, -30, -30, -40]`

(qui compte bien 10 éléments, quand `Temps` et `Position` en comptent 11).

B.2 – Suites implicites : récurrentes

Lorsqu'on a une suite récurrente d'ordre α ($\alpha = 1$ pour une suite récurrente simple, 2 pour une suite récurrente double, etc), il faut α valeurs pour initialiser la suite.

- Suite récurrente simple : il faut initialiser U_0
- Suite récurrente double : il faut initialiser U_0 et U_1
- etc.

Si on reprend notre objectif de vouloir calculer les termes U_0 à U_{N-1} (on veut donc N termes en tout), rangés dans une liste `U`, on voit qu'il va nous falloir $N - \alpha$ itérations. La structure typique est donc une boucle *for*.

Exemple : prenons la suite récurrente définie par $\begin{cases} U_0 = 2 \\ U_n = 3U_{n-1} + 1 \text{ pour } n \geq 1 \end{cases}$. Le code type pour calculer la liste des $(U_n)_{0\leq n\leq N-1}$ est :

```
N = 10
LU = [2] # plutôt que d'initialiser vide, on initialise avec les CI
for n in range(1,N) : # n allant de 1 à N-1, donc N-1 itérations
    Un = 3*LU[n-1] + 1    # ATTENTION, ne pas écrire LU[n] = ... car à
                          # l'itération actuelle LU ne compte que n éléments,
                          # index allant de 0 à n-1…
    LU.append(Un) # c'est là qu'on déclare LU[n], taille de la liste augmente
```

Pour vérifier que c'est bien compris, reprenons l'exemple de la suite de *Fibonacci* déjà vue auparavant. On définit :

$$\begin{cases} U_0 = 1 \ ; \ U_1 = 1 \\ U_n = U_{n-1} + U_{n-2} \text{ pour } n \geq 2 \end{cases}$$

Q11 – Écrire une fonction `Fibonacci(N)` qui renvoie la liste des termes allant de U_0 à U_{N-1} (N termes en tout).

Contrainte : votre fonction devra tenir en 6 lignes, en-tête et return compris.

Aide : on rappelle que pour une liste `L`, `L[-1]` correspond au dernier élément, `L[-2]` l'avant dernier, etc.

Test : `print(Fibonacci(10))  # doit afficher [1, 1, 2, 3, 5, 8, 13, 21, 34, 55]`

Dans le cas (rare) où on s'appuie sur une boucle **while** plutôt qu'une boucle **for**, on ne se pose pas le problème des bornes du *range* (puisqu'il n'y en a pas). Mais l'initialisation doit quand même être faite, en amont de la boucle, comme ce que vous avez fait pour la question **Q8**.

Q12 – ⌨ Sans vous servir de la question précédente, écrire une fonction `Fibo100()` qui ne prend pas d'argument, et renvoie la liste de tous les termes successifs de (U_n) qui soient ≤ 100.

Contrainte : votre fonction devra tenir en 7 lignes, en-tête et return compris.

Test : `print(Fibo100())  # doit afficher [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]`

C / Moyenne glissante

Avant de continuer sur ce dernier bout de TD, allez consulter le document *Outils – Moyenne Glissante*.pptx disponible sur mon site.

Notons `M` un entier, et définissons une fenêtre de taille `2M+1`. Le principe d'une moyenne glissante centrée est d'appliquer à chaque élément « loin des bords » une moyenne sur les `M` voisins de chaque côté, c'est-à-dire, pour une liste $L = (L_i)_{0\leq i\leq N-1}$

$$\forall i \in [\![M, N-1-M]\!], \qquad L_i^{\text{filtr}} = \langle L_k \rangle_{i-M\leq k\leq i+M} = \frac{1}{2M+1}\sum_{k=i-M}^{i+M} L_i$$

Contrairement à ce qui a été vu dans la capsule, on souhaite ici que la liste filtrée (générée par l'algorithme de moyenne glissante) ait la même taille que la liste non filtrée (liste en entrée), nous prendrons pour convention qu'aux bords :

$$\forall i \in [\![0, M-1]\!] \cup [\![N-M, \quad N-1]\!], \qquad L_i^{\text{filtr}} = L_i \quad \text{(bords non filtrés)}$$

Q13 – ⌨ Écrire une fonction `moy_gliss(U, M)` prenant en argument une liste `U` et une taille de demi-fenêtre `M` entière, et renvoyant la liste filtrée avec la méthode décrite précédemment.

Tests : Soit la liste de test suivante `L_tst3 = [0,15,-15,30,15,45,15,60,90,75,60]`

```
print(moy_gliss(L_tst3, 1))    # affiche [0, 0.0, 10.0, 10.0, 30.0, 25.0, 40.0, 55.0,
                                          75.0, 75.0, 60]
print(moy_gliss(L_tst3, 2))    # affiche [0, 15, 9.0, 18.0, 18.0, 33.0, 45.0, 57.0,
                                          60.0, 75, 60]
```

Q13* – ⌨ Petit challenge supplémentaire pour ceux qui voudront (assez facile) : ajouter la contrainte de s'interdire l'utilisation de toute structure conditionnelle **if**. Le **while** est interdit aussi, d'ailleurs.

Indice : penser extraction et concaténation de listes… ! Il suffit d'extraire les *M* premiers et les *M* derniers éléments de *U* (qui ne sont pas filtrés), et les concaténer au début et à la fin de la liste à filtrée à renvoyer.