Séance Python — Préparation à l'oral Agro/Véto et de G2E

Ex 1 :
1) Écrire une fonction `melange(n)` qui prend en entrée un entier n et renvoie une permutation aléatoire de $[\![0,\, n-1]\!]$ selon le procédé suivant :
   - créer la liste $L = [0, 1, \ldots, n-1]$;
   - pour i variant de 0 à $n-2$, choisir j uniformément au hasard dans $[\![i,\, n-1]\!]$, puis échanger $L[i]$ et $L[j]$.
2) Écrire `melange_mot(mot)` qui renvoie les lettres d'une chaîne mélangées aléatoirement (en utilisant `melange`).
3) *(Bonus.)* Écrire `melange_texte(texte)` qui renvoie le texte avec les mots mélangés aléatoirement. *Exemple :* `"je sais bien"` $\longrightarrow$ `"sais bien je"`.

Ex 2 : Un dérangement de $[\![0,\, n-1]\!]$ est une permutation dans laquelle aucun élément i n'est en position i.

1) Écrire `Permutation(n)` renvoyant une permutation aléatoire de $[\![0,\, n-1]\!]$.
2) Écrire `estUnDerangement(Perm)` renvoyant `True` si et seulement si `Perm` est un dérangement.
3) Écrire `Derangement(n)` renvoyant un Derangement de $[\![0,\, n-1]\!]$ par la méthode du rejet.
4) *(Discussion.)* Estimer la probabilité qu'une permutation aléatoire soit un dérangement pour n grand. La méthode du rejet est-elle efficace ? Proposer une alternative.

Ex 3 :
1) Écrire une fonction `secondmax(L)` qui renvoie le deuxième maximum d'une liste L, sans modifier L ni créer de nouvelle liste.
2) Écrire une fonction `position(L, x)` qui renvoie la position qu'occuperait l'élément $x \in L$ si L était triée par ordre croissant, sans modifier L ni créer de nouvelle liste.

Ex 4 : On dispose de la fonction `ord(c)` qui renvoie le code ASCII du caractère `c`.

1) Écrire une fonction `code(motif)` qui prend en paramètre une chaîne de caractères `motif` et renvoie la somme des codes de ses caractères.
2) Écrire une fonction `liste_codes(texte, p)` qui prend en paramètre une chaîne `texte` et un entier $p \geqslant 1$, et renvoie la liste des codes de tous les sous-motifs de longueur p du texte (par fenêtre glissante). *Exemple : pour* `texte = 'GACTAAG'` *et* `p = 3`*, renvoyer* `[code('GAC'), code('ACT'), code('CTA'), code('TAA'), code('AAG')]`.
3) *(Bonus.)* Réécrire `liste_codes` par mise à jour incrémentale : calculer le code du premier motif, puis, à chaque glissement, soustraire le code du caractère sortant et ajouter le code du caractère entrant. Comparer l'efficacité de chaque version ?

Ex 5 : Écrire une fonction `election1(L)` prenant en argument une liste L contenant un ou deux mots différents (éventuellement répétés), et renvoyant le mot qui apparaît le plus de fois dans L. Si aucun mot n'est plus fréquent que l'autre, la fonction renvoie `None`.

Exemple : pour $L =$ [`"ombre"`, `"marbre"`, `"marbre"`, `"ombre"`, `"ombre"`]*, la fonction renvoie* `"ombre"`.

1) Écrire une première version de `election1` utilisant un parcours de liste et un compteur.
2) Peut-on généraliser à une liste contenant un nombre quelconque de mots différents ?

Ex 6 : Un cours est représenté par un tuple `s` tel que `s[0]` est l'heure de début et `s[1]` est l'heure de fin (avec `s[0] < s[1]`).

1) Écrire une fonction `appartient(s, m)` prenant en argument un cours `s` et un réel `m`, et renvoyant `True` si m appartient au créneau [`s[0]`, `s[1]`], `False` sinon.

   Exemples : `appartient((1,3), 2)` *renvoie* `True` *;* `appartient((1,3), 4)` *renvoie* `False`.

2) Écrire une fonction `intersecte(c1, c2)` prenant en argument deux cours `c1` et `c2` et renvoyant `True` si les deux cours se chevauchent, `False` sinon.

3) On dispose d'une liste `L` de cours (une liste de tuples) correspondant aux horaires des cours d'une salle. Écrire une fonction `liste_intersecte(L, c)` prenant en argument `L` et un cours `c`, et renvoyant l'indice de la première salle disponible dans laquelle le cours `c` peut se dérouler, et -1 si aucune salle n'est disponible.

Ex 7 : Une séquence d'ADN est représentée par une liste de caractères parmi `'a'`, `'t'`, `'c'`, `'g'`.

1) On dispose du programme suivant. Décrire ce qu'il fait.

```
bases = ['a', 't', 'c', 'g']
def freq_bases(seq):
    compteurs = [0, 0, 0, 0]
    for b in seq:
        for i in range(4):
            if b == bases[i]:
                compteurs[i] += 1
    return compteurs
```

2) Écrire une fonction `premiere_repetition(seq)` prenant en argument une liste de caractères `seq` et renvoyant un tuple `(c, k)` où `c` est le premier caractère de `seq` et `k` est le nombre de fois qu'il apparaît consécutivement au début de `seq` (sans interruption par un autre caractère).

Exemple : pour `seq = ['c','c','c','t','t','a','c']`, la fonction renvoie `('c', 3)`.

3) Écrire une fonction `toutes_repetitions(seq)` qui renvoie la liste de tous les tuples `(c, k)` correspondant aux runs (séquences consécutives d'un même caractère) dans `seq`, dans l'ordre d'apparition.

Exemple : pour `seq = ['c','c','c','t','a','g','t','t','a','c']`, la fonction renvoie : `[('c',3), ('t',1), ('a',1), ('g',1), ('t',2), ('a',1), ('c',1)]`.

4) Écrire une fonction `decompress(runs)` prenant en argument une liste de tuples `[(c1,k1),(c2,k2),` et renvoyant la séquence originale (liste de caractères). Vérifier que `decompress(toutes_repetiti` `== seq`.

(Discussion.) Quelle est la complexité de `toutes_repetitions` ? Dans quel cas ce codage par runs compresse-t-il efficacement la séquence ?

Ex 8 : Un polynôme $P = a_0 + a_1X + \cdots + a_nX^n$ est représenté par la liste de ses coefficients $[a_0, a_1, \ldots, a_n]$, du coefficient constant au coefficient dominant.

Exemple : $P(X) = 2 + 4X + 5X^2$ est représenté par `[2, 4, 5]`.

1) Écrire une fonction `evalue(P, alpha)` prenant en argument la liste `P` et un réel `alpha`, et renvoyant $P(\alpha)$.

2) On donne l'algorithme suivant, exprimé en français :

Poser $y = a_n$.
Pour k variant de $n-1$ à 0 (en décroissant) :
    Poser $y = y \cdot \alpha + a_k$.
Renvoyer y.

Retranscrire cet algorithme en une fonction Python `horner(P, alpha)`.

3) Vérifier les deux programmes sur l'exemple $P(X) = 2 + 4X + 5X^2$ et $\alpha = 3$.

4) *(Discussion.)* Comparer `evalue` et `horner` en nombre de multiplications.

Ex 9 : On définit la fonction S sur les entiers naturels : $S(p)$ est la somme des carrés des chiffres de p. *Exemple :* $S(450) = 4^2 + 5^2 + 0^2 = 41$.

1) Écrire une fonction `S(p)` prenant en argument un entier p et renvoyant la somme des carrés de ses chiffres.
2) On définit la suite $t_0 = n$, $t_{k+1} = S(t_k)$. On admet que cette suite finit toujours par atteindre 1 (auquel cas n est dit heureux) ou par entrer dans un cycle contenant 89 (auquel cas n n'est pas heureux).
   Écrire une fonction `est_heureux(n)` renvoyant `True` si n est heureux et `False` sinon.
3) Écrire une fonction `nb_heureux(n)` prenant en argument un entier n et renvoyant le nombre d'entiers heureux dans $[\![1, n]\!]$.
4) *(Discussion.)* Combien d'entiers heureux y a-t-il parmi $[\![1, 100]\!]$? Proposer une amélioration de `est_heureux` pour éviter de recalculer des valeurs déjà connues (en utilisant une liste de valeurs déjà testées).

Ex 10 : On considère la suite définie par : u_0 donné, $\quad u_{n+1} = u_n(1 - u_n)$.

1) Écrire une fonction `suite_log(u0, n)` prenant en argument u_0 et n, et renvoyant la liste des n premiers termes $(u_0, u_1, \dots, u_{n-1})$ de cette suite.
2) Écrire une fonction `premier_inferieur(u0, e)` prenant en argument u_0 et un réel $e > 0$, et renvoyant l'indice du premier terme de la suite strictement inférieur à e.
3) On pose $V_n = n\,u_n$. Écrire une fonction `croissance_Vn(u0)` qui renvoie la chaîne `"La suite (Vn) est croissante"` si (V_n) est croissante sur les 1000 premiers termes, et sinon l'indice du premier terme pour lequel $V_{n+1} < V_n$.

Ex 11 :

1) Écrire une fonction `recherche(x, L)` prenant en argument un élément `x` et une liste `L`, et renvoyant la liste des indices auxquels `x` apparaît dans `L`.
   Exemple : pour `L = [1,4,1,5,6,1]`, `recherche(1, L)` *renvoie* `[0, 2, 5]`.
2) Écrire une fonction `elements_distincts(L)` renvoyant la liste des éléments distincts de `L`, dans l'ordre de leur première apparition, sans utiliser les fonctions `set` ou `sorted`.
3) Écrire une fonction `table(L)` prenant en argument une liste `L` et renvoyant un tableau d'effectifs sous la forme d'une liste de deux listes : la première contenant les éléments distincts de `L`, la seconde contenant leurs effectifs respectifs.
   Exemple : pour `L = [1,5,1,1,4,3,9,1]`, `table(L)` *renvoie* `[[1,5,4,3,9], [4,1,1,1,1]]`.

Ex 12 : Un polynôme $P = a_0 + a_1X + \cdots + a_nX^n$ est représenté par la liste de ses coefficients dans l'ordre croissant des degrés : $[a_0, \dots, a_n]$. *Exemple :* $3X^2 + 4X - 7$ *est représenté par* `[-7, 4, 3]`.

1) Écrire une fonction `evalue(P, x)` prenant en argument une liste de coefficients `P` et un réel `x`, et renvoyant la valeur $P(x)$.
2) Écrire une fonction `somme(P, Q)` renvoyant la liste des coefficients de $P + Q$.
   a. Dans le cas où `P` et `Q` sont de même longueur.
   b. Dans le cas général où `P` et `Q` peuvent être de longueurs différentes.
      (On complétera le polynôme le plus court par des zéros.)
3) Écrire une fonction `termes_non_nuls(P)` renvoyant la liste des couples `(coefficient, degré)` pour les termes de coefficient non nul de P, triés par degré croissant.
   Exemple : pour $P = -7 + 3X + 2X^5 + 3X^{10}$, *la fonction renvoie* `[(-7,0),(3,1),(2,5),(3,10)]`.
4) Écrire une fonction `egal(P, Q)` renvoyant `True` si les polynômes P et Q sont identiques (mêmes coefficients), `False` sinon.

On rappelle que la série exponentielle tronquée à l'ordre n est : $e_n(x) = \sum_{i=0}^{n} \frac{x^i}{i!}$.

5) Écrire une fonction `factorielle(n)` calculant $n!$ de façon itérative.

6) Écrire une fonction itérative `exponentielle(x, n)` renvoyant $e_n(x)$.

7) Écrire une version récursive `exponentielle_rec(x, n)` de la même fonction.

Ex 13 :

1) Écrire une fonction `somme(L, i, j)` prenant en argument une liste d'entiers `L` et deux indices `i` et `j`, et renvoyant la somme des éléments de `L` d'indice compris entre `i` et `j` inclus. *Exemple : `somme([1,3,8,4,7], 1, 3)` renvoie* $3 + 8 + 4 = 15$.

2) Écrire une fonction `somme_max(L, k)` prenant en argument une liste `L` et un entier $k \geqslant 1$, et renvoyant la somme maximale de k éléments consécutifs de `L`, en utilisant `somme`.
Exemple : pour `L = [1,3,8,4,7]` et `k = 2`, les sommes de 2 éléments consécutifs sont $4, 11, 12, 11$, *donc la fonction renvoie* 12.

3) *(Discussion.)* Proposer une version plus efficace utilisant une mise à jour incrémentale de la somme courante. *(Observer que `somme(L,i+1,i+k)` = `somme(L,i,i+k-1)` - `L[i]`+`L[i+k]`.)*

Ex 14 :

1) Écrire une fonction `dans_intervalle(L, a, b)` prenant en argument une liste `L` et deux réels `a` et `b`, et renvoyant `True` si toutes les valeurs de `L` sont dans $[a, b]$, `False` sinon.

2) On suppose désormais que a et b sont des entiers et que toutes les valeurs de `L` sont des entiers appartenant à $[\![a, b]\!]$. Écrire une fonction `comptage(L, a, b)` renvoyant une liste `C` de longueur $b - a + 1$ telle que `C[i]` est le nombre d'occurrences de la valeur $a + i$ dans `L`, en utilisant une seule boucle.
Exemple : pour `L = [3,1,2,1,3,3]`, `a = 1`, `b = 3`, la fonction renvoie `[2, 1, 3]` (2 fois le 1, 1 fois le 2, 3 fois le 3).

3) À l'aide de `comptage`, écrire une fonction `tri_comptage(L, a, b)` qui renvoie la liste `L` triée par ordre croissant.

Ex 15 :

1) Écrire une fonction `long_valide(L, n)` prenant en argument une liste `L` et un entier n et renvoyant `True` si la longueur de `L` est comprise entre 0 et n (au sens large), `False` sinon.

2) Écrire une fonction `est_permutation(L)` prenant en argument une liste `L` et renvoyant `True` si `L` est une permutation de $[\![0, n-1]\!]$ (où n = `len(L)`), c'est-à-dire si `L` contient exactement une fois chaque entier de $[\![0, n-1]\!]$.

3) *(Bonus.)* Écrire une fonction `composee(sigma, tau)` prenant en argument deux permutations (listes de même longueur) et renvoyant leur composée $\sigma \circ \tau$, c'est-à-dire la liste `[sigma[tau[i]] for i in range(n)]`.

Ex 16 :

1) Écrire une fonction `non_multiples(L, k)` prenant en argument une liste `L` et un entier $k \geqslant 2$, et renvoyant la liste des éléments de `L` qui ne sont ni 0, ni 1, ni multiples de k (en conservant 0 et 1 dans la liste). *(Ne pas modifier `L` ; construire une nouvelle liste.)*

2) Écrire une fonction `mise_a_zero(L, k)` prenant en argument une liste `L` et un entier $k \geqslant 2$, et remplaçant par 0 dans `L` tous les multiples de k strictement supérieurs à k.

3) Écrire une fonction `crible(n)` qui :

- initialise la liste $L = [0, 1, 2, \ldots, n]$;
- pour chaque entier k de 2 à $\lfloor \sqrt{n} \rfloor$, appelle `mise_a_zero(L, k)` ;
- renvoie la liste des éléments non nuls et différents de 1, c'est-à-dire la liste des nombres premiers inférieurs ou égaux à n.