Correction Séance Python _ 2

Préparation à l'oral Agro/Véto et G2E

BCPST 2A 2025/2026

Exercice 1 : Mélange aléatoire

Question 1 — `melange(n)`

Il suffit de suivre l'algorithme décrit dans l'énoncé.

```
import random as rd

def melange(n):
    L = [ k for k in range(n) ]
    for i in range(n - 1):
        j = rd.randint(i, n - 1)
        L[i], L[j] = L[j], L[i]
    return L
```

Question 2 — `melange_mot(mot)`

On obtient une permutation aléatoire des indices via `melange`, puis on reconstruit la chaîne dans cet ordre dans une autre variable.

On ne peut pas modifier une chaine de caractère donc on la reconstruit à partir d'une chaine vide.

```
def melange_mot(mot):
    n = len(mot)
    perm = melange(n)
    sortie = ''
    for k in range(n);
        sortie += mot[Perm[k]]
    return sortie
```

Question 3 (Bonus) — `melange_texte(texte)`

On découpe le texte en mots, on mélange la liste de mots, puis on la réassemble.

```
def melange_texte(texte):
    """ On suppose qu'il n'y a jamais des double-espace"""
    L = []
    mot = ''
    for c in texte:
        if c != ' ':
            mot += c
        else:
            L.append(mot)
            mot = ''
    L.append(mot)                    # Ici L contient la liste des mots de texte.

    n = len(L)
    sortie = ''
    M = melange(n)
    for i in M:
        sortie += L[i] + ' '         # On ajoute des espaces entre deux mots.
    return sortie
```

Exercice 2 : Dérangements

Question 1 — `Permutation(n)`

```
def Permutation(n):
    L = [ k for k in range(n) ]
    S = []
    for i in range(n):
        j = rd.randrange(len(L))
        S.append(L.pop(j))
    return S
```

Question 2 — `estUnDerangement(Perm)`

Exercice très classique : attention à la position des `return True` et `return False`

```
def estUnDerangement(Perm):
    for i in range(len(Perm)):
        if Perm[i] == i:
            return False
    return True
```

Question 3 — `Derangement(n)` par rejet

On tire des permutations aléatoires jusqu'à en obtenir un dérangement. *On rejette les permutations qui ne conviennent pas.*

```
def Derangement(n):
    P = Permutation(n)
    while not estUnDerangement(P):
        P = Permutation(n)
    return P
```

Une deuxième version.

```
def Derangement(n):
    while True:
        P = Permutation(n)
        if estUnDerangement(P):
            return P
```

Une version récursive inspirée par une idée de Melvin.

```
def Derangement(n):
    Perm = Permutation(n)
    if estUnDerangement(Perm):
        return Perm
    return Derangement(n)
```

Exercice 3 : Second maximum et position

Question 1 — `secondmax(L)`

On parcourt L en maintenant le maximum `m1` et le second maximum `m2`.

```
def secondmax(L):
    max1, max2 = L[0], L[1]
    if max2 > max1:
        max1, max2 = max2, max1
    for i in range(2, len(L)):
        if L[i] > max1:
            max2, max1 = max1, L[i]
        elif max2 < L[i] < max1:
            max2 = L[i]
    return max2
```

Une deuxième version.

```
inf = float('inf')

def secondmax(L):
    max1 = -inf
    max2 = -inf
    for x in L:
        if x > max1:
            max2 = max1
            max1 = x
        elif max2< x < max1:
            max2 = x
    return max2
```

Question 2 — `position(L, x)`

La position de x dans L triée par ordre croissant est le nombre d'éléments de L strictement inférieurs à x.

```
def position(L, x):
    count = 0
    for elem in L:
        if elem < x:
            count += 1
    return count
```

Remarque : cette fonction ne modifie pas L et ne crée pas de nouvelle liste (on utilise seulement un compteur entier).

Exercice 4 : Codes de Rabin-Karp

Question 1 — `code(motif)`

```
def code(motif):
    s = 0
    for c in motif:
        s += ord(c)
    return s
```

Question 2 — `liste_codes(texte, p)`

```
def liste_codes(texte, p):
    codes = []
    for i in range(len(texte) - p + 1):
        codes.append(code(texte[i:i+p]))
    return codes
```

Pour `texte = 'GACTAAG'` et `p = 3`, on obtient les codes de `'GAC'`, `'ACT'`, `'CTA'`, `'TAA'`, `'AAG'`.

Question 3 (Bonus) — Version incrémentale

Au lieu de recalculer le code de chaque fenêtre depuis zéro, on met à jour le code courant en soustrayant le caractère sortant et en ajoutant le caractère entrant.

```
def liste_codes_incr(texte, p):
    if len(texte) < p:
        return []
    c_courant = code(texte[:p])
    codes = [c_courant]
    for i in range(1, len(texte) - p + 1):
        c_courant -= ord(texte[i - 1])
        c_courant += ord(texte[i + p - 1])
        codes.append(c_courant)
    return codes
```

Exercice 5 : Élection

Question 1 — Version avec compteur

La liste ne contient qu'un ou deux mots distincts.

```
def election1(L):
    mot_1 = L[0]
    count = 0
    for mot in L:
        if mot == mot_1:
            count += 1
        else:
            count -= 1
            mot_2 = mot
    if count == 0:
        return None
    elif count > 0:
        return mot_1
    return mot_2
```

Question 2 — Généralisation à plusieurs mots

Le plus simple est d'utiliser des dictionnaires mais on va faire autrement car on ne les manipule pas souvent.

```
def trouve_indice(valeur, liste):
    for i in range(len(liste)):
        if liste[i] == valeur:
            return i
    return None

def ind_max(L):
    p = [0]
    maxi = L[0]
    for i in range(1,len(L)):
        if L[i] > maxi:
            maxi = L[i]
            p = [i]
        elif L[i] == maxi:
            p.append(i)
    return p

def election_generale(L):
    nom = [L[0]]
    votes = [1]
    for k in range(1, len(L)):
        if L[k] in nom:
            i = trouve_indice(L[k], nom)
            votes[i] += 1
        else:
            nom.append(L[k])
            votes.append(1)
    premier = ind_max(votes)
    if len(premier) > 1:
        return None
    return nom[premier[0]]
```