

Partie I : Les ensembles finis usuels.

1. On considère le programme suivant :

```
def liste(p, n):
    L = [[]]
    for _ in range(p):
        L = [x + [k] for x in L for k in range(n)]
    return L
```

- (a) Tester la fonction avec plusieurs couple (p, n) et observer le résultat obtenu.
 - (b) Que fait ce programme ? Comment évolue la variable L ?
 - (c) Combien d'éléments contient `liste(p, n)` ?
2. (a) Écrire une fonction `test(L)` prenant pour argument L une liste et qui renvoie `True` si et seulement si :
- le dernier élément de L vaut 0,
 - et L contient exactement trois zéros.
- (b) À l'aide de la fonction `liste(p, n)` précédente, compter le nombre de listes de longueur 7 formées d'éléments de $\{0, 1, 2, 3\}$ vérifiant ce test.
 - (c) À l'aide d'un raisonnement, compter le nombre de listes de longueur p formées d'éléments de $\{0, \dots, n-1\}$ vérifiant ce test. (*controler cette formule dans l'exemple précédent*)
3. On considère les deux programmes suivants :

```
def fonction_2(p, n):
    L = [[]]
    for _ in range(p):
        L = [x + [k] for x in L for k in range(n) if k not in x]
    return L
```

```
def fonction_3(p, n):
    L = [[k] for k in range(n)]
    for _ in range(p-1):
        L = [x + [k] for x in L for k in range(x[-1]+1, n)]
    return L
```

- (a) Que font ces deux fonctions ?
 - (b) Quelle est la longueur de `fonction_2(p, n)` ? de `fonction_3(p, n)` ?
4. On a déjà vu le programme suivant :

```
def fonction_4(mot):
    L = ['']
    for _ in range(len(mot)):
        L = [x + k for x in L for k in set(mot)
              if x.count(k) < mot.count(k)]
    return L
```

- (a) Que fait cette fonction ?
- (b) Quelle est la longueur de `fonction_4('MISSISSIPPI')` ?
- (c) Retrouver le résultat précédent par un raisonnement.

Partie II. Opérations sur les ensembles finis.

Les ensembles sont représentés par des listes d'éléments deux à deux distincts.

(Cette représentation est unique à l'ordre près des éléments)

Dans un premier temps, on n'utilisera pas le type `set`.

1. (a) Écrire une fonction `distinct2a2(L)` qui renvoie `True` si tous les éléments de L sont distincts deux à deux.
- (b) Tester la fonction avec : $L = [2, 3, 5, 6, 3]$ et $L = [2, 3, 5, 6, 1]$

Dans la suite, toutes les listes données en argument ont des éléments deux à deux distincts.

2. (a) Écrire une fonction `union(L1, L2)` qui renvoie la réunion de deux listes.
(b) Écrire une fonction `intersection(L1, L2)` qui renvoie leur intersection.
(c) Écrire une fonction `disjointes(L1, L2)` qui teste si deux listes sont disjointes.
(d) Tester vos fonctions avec par exemple : `L1 = [2, 3]` et `L2 = [1, 3]`
3. (a) Écrire une fonction `disjointes2a2(Liste)` qui teste si une liste de listes est formée d'ensembles disjoints deux à deux.
(b) Tester par exemple avec : `Liste = [[1, 3], [2], [0, 4, 5, 6], [7, 8]]`
puis avec : `Liste = [[1, 3], [2], [0, 4, 5, 6], [7, 8, 1]]`
4. (a) Écrire une fonction `inclus(A, B)` qui teste si la liste `A` est incluse dans la liste `B`.
(b) Écrire une fonction `partition(B, Famille)` qui vérifie si la famille `Famille` forme une partition de `B`, c'est-à-dire :
 - les sous-ensembles de `famille` sont disjoints deux à deux ;
 - la réunion de ses sous-ensembles est égale à `B`.
(c) Tester par exemple avec : `B = [1, 2, 5, 6, 3]` et `Famille = [[3], [1, 2], [5, 6]]`
5. Reprendre les fonctions précédentes (2), 3) et 4)) en utilisant le type `set`.

Partie III. Des exemples sortis du concours.

1. (a) Ecrire une fonction `valeur(L, n)` qui prend en entrée `L` une liste d'entiers et `n` un entier et qui renvoie si oui ou non `L` contient des entiers entre $\llbracket 0; n - 1 \rrbracket$.
(b) Ecrire une fonction `permutation(L)` qui prend en entrée `L` une liste d'entiers et qui renvoie si oui ou non `L` est une permutation de $\llbracket 0; n - 1 \rrbracket$ où `n` est la longueur de `L`.
2. Une permutation de $\llbracket 0, n - 1 \rrbracket$ est une liste de longueur `n` contenant exactement une fois chaque élément de $\llbracket 0, n - 1 \rrbracket$. Un dérangement est une permutation pour lequel tout élément `i` n'est pas en position `i`.
(a) Écrire une fonction `Permutation`, prenant en argument `n` et renvoyant au hasard une permutation de $\llbracket 0, n - 1 \rrbracket$
(b) Écrire une fonction `estUnArrangement` prenant en argument une permutation `Perm`, renvoyant `True` si `Perm` est un dérangement, `False` sinon
(c) Écrire une fonction `Dérangement`, prenant en argument `n` et renvoyant un dérangement de $\llbracket 0, n - 1 \rrbracket$ en utilisant les deux fonctions précédentes
Ensuite discussion pour essayer d'écrire une fonction plus efficace, qui n'utiliserait pas les fonctions précédentes
3. On considère une liste `L` dont les valeurs sont prises dans l'intervalle d'entiers $\llbracket 0; k - 1 \rrbracket$.
(a) Écrire une fonction `occurrence(L, k)` qui prend en entrée une liste `L` et un entier `k` et qui renvoie la liste des nombres d'occurrences des entiers de 0 à `k-1` dans la liste `L`
On supposera que `L` est une liste de nombres et que $0 \leq k < \text{len}(L)$, la fonction n'aura pas à le vérifier.
(b) Si la fonction précédente n'a pas été faite avec une seule boucle, la refaire avec une seule boucle, sinon passer à la question suivante.
(c) Écrire en utilisant la fonction `occurrence(L, k)` une fonction `tri_denombre(L)` qui trie dans l'ordre croissant la liste `L` passée en argument.
4. (a) Ecrire une fonction `palindrome(mot)` qui prend pour paramètre `mot` une chaîne de caractères et qui renvoie `True` si `mot` est un palindrome et `False` sinon.
(b) Ecrire une fonction `anagramme(mot1, mot2)` qui prend pour paramètres `mot1` et `mot2` deux chaînes de caractères et qui renvoie `True` si `mot1` est une anagramme de `mot2` et `False` sinon.
(c) Ecrire une fonction `liste_anagrammes(mot)` qui prend pour paramètre `mot` une chaîne de caractères et qui renvoie la liste de toutes les anagrammes de `mot`.

Précision :

Un **palindrome** est un mot qui reste le même si on le lit de gauche à droite ou de droite à gauche.

Exemples de palindromes : radar, kayak, ici, ressasser, AABBA.

Une **anagramme** est un mot obtenu en permutant les lettres d'un mot.

Ici les mots sont justes des chaînes de caractères, ils n'ont pas à avoir de sens.

Par exemple, la liste des 12 anagrammes de ANNE est :

AENN , ANEN, ANNE, EANN, ENAN, ENNA, NAEN, NANE, NEAN, NENA, NNAE, NNEA