FSAC
FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

TD 2 : STRUCTURES DE DONNEES

Partie I : Les listes

Exercice 1 : Les polynômes

Un polynôme sera représenté par une liste, les valeurs sont les coefficients et les indices sont les poids.

Exemple : le polynôme suivant p(x) = $4x^3$+ 2x + 6 sera représenté par la liste : P = [6, 2, 0, 4]

1. (*) Ecrire une fonction **poly(p, x)** qui calcule la valeur numérique du polynôme P pour x. (Vous pouvez utiliser un algorithme simple qui utilise la puissance ou la méthode de Horner)

   $$P(x) = (((a_n x + a_{n-1})x + a_{n-2})x + ... + a_1)x + a_0$$

   Exemple : poly(P, 2) = 42

2. (*) Ecrire une fonction **somme(a, b)** qui calcule et retourne la somme de deux polynômes.

   Exemple : pour les deux polynômes P = [5, 3, 2] (Représente $2x^2 + 3x + 5$) et Q = [4, 0, 1, 6] (Représente $6x^3 + 1x^2 + 4$) somme(P, Q) retourne [9, 3, 3, 6]

3. (**) Ecrire une fonction **produit(a, b)** qui calcule et retourne le produit de deux polynômes.

   Exemple : Pour P=[2, 0, 1, 2] et Q=[1, 2, 4] la produit(P,Q) retourne [2, 4, 9, 4, 8, 8]

Correction :

```
def poly(P, x):
    s = 0
    for i in range(len(P)):
        s += P[i]*x**i
    return s

def polyV2(p, x):
    s = 0
    for i in range(len(p)-1, -1 ,-1):
        s = s * x + p[i]
    return s

def somme(a, b):
    s = []
    mn = min(len(a), len(b))
    for i in range(mn):
        s.append(a[i] + b[i])
    s.extends(a[mn:] + b[mn:])
    return s

def produit(a, b):
    da, db = len(a)-1 , len(b) - 1
```

```
    dp = da + db
    P = [0]*(dp+1)
    for i in range(len(a)) :
        for j in range(len(b)) :
            P[i+j] += a[i]*b[j]
    return P
```

Exercice N° 2 : Opération sur les ensembles mathématiques

Un ensemble mathématique est un groupement d'objets distincts, appelés éléments de cet ensemble. La théorie des ensembles est l'étude des propriétés et des opérations sur des ensembles (appartenance, inclusion, réunion, . . .). Elle représente une branche essentielle des mathématiques.

1. Ecrire une fonction **union(L1, L2)** qui prend en argument deux listes **L1 et L2** d'entiers et renvoie l'union de L1 et L2.
2. Ecrire une fonction **intersection(L1, L2)** qui prend en argument deux listes **L1 et L2** d'entiers et renvoie l'intersection de L1 et L2.

Correction :

```
def union(L1, L2):
    L = L1[:]
    for a in L2 :
        if a not in L1 :
            L.append(a)
    return L

def intersection(L1, L2):
    L = []
    for a in L1 :
        if a in L2 :
            L.append(a)
    return L
```

Exercice 3 : Statistiques

1. (*) Ecrire une fonction, **moyenne(L)** qui retourne la moyenne de la liste L.
2. (**) Ecrire une fonction, **mediane(L)** qui retourne la médiane de la liste L.
3. (**) Ecrire une fonction, **mode(L)** qui retourne les modes de la liste L.
4. (***) Ecrire une fonction, **decomposer(n)** qui décompose un entier positif en nombres premiers. *Exemple :* decomposer(120) retourne [2, 2, 2, 3, 5]

Correction :

```
def moyenne(L):
    return sum(L)/len(L)

def moyenneV2(L):
    S = 0
    for a in L :
```

```
        S+= a
    return S/len(L)

def mediane(L):
    A = sorted(L)
    if len(A)%2 == 1 :
        return A[len(A)//2]
    return (A[len(A)//2]+A[len(A)//2-1])/2

def mode(L):
    max_occ = 0
    modes = []
    for a in L :
        m = L.count(a)
        if m > max_occ :
            max_occ = m
            modes = [a]
        elif m==max_occ and a not in L:
            modes.append(a)

    return modes


def decompsoe(n) :
    L = []
    x = 2
    while n!=1 :
        while n%x == 0 :
            L.append(x)
            n = n // x
        x += 1
    return L
```

Exercice N° 4 : Compte

On considère un tableau de bits dont les éléments sont égaux aux entiers 0 ou 1. Rédiger en Python une fonction « **compte(L)** » calculant le nombre maximal de 0 consécutifs présents dans ce tableau. Par exemple, pour le tableau suivant la fonction devra retourner la valeur 6 :

1	0	1	0	0	0	0	0	0	1

Correction :

```
def compte(L):
    max_zeros = 0
    nb_zeros = 0

    for x in L:
```

```
        if x == 0:
            nb_zeros += 1
            if nb_zeros > max_zeros:
                max_zeros = nb_zeros
        else:
            nb_zeros = 0

    return max_zeros
```

Partie II : Les chaînes de caractères

1. (*) Implémentez une fonction **est_palindrome(s)** qui vérifie si la chaîne s est un palindrome, c'est-à-dire qu'elle se lit de la même manière dans les deux sens.
2. (*) Implémentez une fonction **anagrams(s1, s2)** qui détermine si deux chaînes s1 et s2 sont des anagrammes, c'est-à-dire qu'elles contiennent les mêmes lettres avec la même fréquence, mais dans un ordre différent.
3. Ecrire une fonction **conjuguer(verbe)** qui affiche la conjugaison d'un verbe au premier groupe au présent de l'indicatif.
4. (***) Écrivez une fonction **compresser (s)** qui prend une chaîne s composée de lettres répétées et la compresse en une forme plus concise. Par exemple, "aaabbcccc" deviendrait "a3b2c4".

Correction :

```
def est_palindrome(s):
  s = s.replace(" ", "")
  s = s.upper()
  r = s[::-1]
  return s==r

def anagrams(s1, s2):
  s1 = sorted(s1)
  s2 = sorted(s2)
  return s1 == s2

def anagrams2(s1, s2):
  if len(s1)!=len(s2):
    return False
  for a in s1:
    if s1.count(a)!=s2.count(a) :
      return False
  return True
```

```python
def conjuguer(verbe):
  assert verbe[-2:]=="er", 'Le verbe doit se terminer par er'
  S = ["Je", "Tu", "Il/Elle", "Nous", "Vous", "Ils/Elles"]
  T = ["e", "es", "e", "ons", "ez", "ent"]
  radical = verbe[:len(verbe)-2]
  for i in range(len(S)) :
    print(S[i]+" "+radical+T[i])

def compresser(ch):
  c = 1
  res = ""
  for i in range(1, len(ch)):
    if ch[i] != ch[i-1]:
      res += ch[i-1]+str(c)
      c = 1
    else :
      c += 1
  res += ch[-1] + str(c)
  return res
```

Partie III : Les matrices

Exercice N° 1 :

1. Ecrire une fonction « unitaire(n) » qui construit et retourne une matrice carrée unitaire de dimension n. Une matrice unitaire est une matrice, telle que :

$$u_{ij} = \begin{cases} 1 \text{ si } i=j \\ 0 \text{ si } i \neq j \end{cases}$$

Ecrire les fonctions utilitaires suivantes :

2. **transopose(M)** qui retourne la transposition M^t d'une matrice M.
3. **somme(A, B)** qui retourne la somme de deux matrices A et B de mêmes dimensions.

En multipliant une matrice A de dimensions M et N avec une matrice B de dimensions N et P on obtient une matrice C de dimensions M et P: **A(M,N) * B(N,P) = C(M,P)**

La multiplication de deux matrices se fait en multipliant les composantes des deux matrices lignes par colonnes :

$$c_{ij} = \sum_{k=1}^{k=N} (a_{ik} * b_{kj})$$

4. **produit(A, B)** » qui retourne le produit des deux matrices A et B.

5. **trace(A)** d'argument A une matrice et qui retourne la somme des coefficients diagonaux de A (appelée trace de A) si A est une matrice carrée et la chaîne de caractères « trace of a non square matrix is not defined » sinon.

6. **is_upper_triangular(A)** qui retourne True si A est triangulaire supérieure.

Correction :

```
def unitaire(n):
  # créer une matrice nulle
  I = [ [0]*n for i in range(n) ]
  for i in range(n):
      I[i][i] = 1
  return I

def transpose(M):
  l = len(M)
  c = len(M[0])
  T = [ [0]*l for i in range(c)  ]
  for i in range(c):
    for j in range(l):
      T[i][j] = M[j][i]
  return T

def somme(A, B):
  l = len(A)
  c = len(A[0])
  S = [ [0]*c for i in range(l)]
  for i in range(l):
    for j in range(c):
      S[i][j] = A[i][j] + B[i][j]
  return S

def produit(A, B):
  assert len(A[0])==len(B), "Matrices invalide"
  m = len(A)
  n = len(A[0])
  p = len(B[0])
  R = [[0]*p for i in range(m)]
  for i in range(m):
    for j in range(p) :
      for k in range(n):
        R[i][j] += A[i][k]*B[k][j]
  return R

def trace(A):
  l = len(A)
  c = len(A[0])
  assert l==c, "trace of a non square matrix is not defined"
  s = 0
```

```
  for i in range(l):
    s += A[i][i]
  return s

def is_upper_triangular(M) :
  n = len(M)
  for i in range(1, n):
    for j in range(0, i):
      if M[i][j] != 0 :
        return False
  return True
```