TD 7 - Énoncés

Exercice 1 :

Calculs sur les vecteurs en dimension 3. Écrire des programmes permettant de :

Saisir un vecteur de réels puis l'afficher à l'envers
Saisir deux vecteurs et afficher leur somme ainsi que leur produit scalaire
Saisir un vecteur et afficher sa norme, normaliser le vecteur et l'afficher
Saisir deux vecteurs et afficher le vecteur produit vectoriel

On utilisera des tableaux et des boucles `pour`

Rappels :

Soient 2 vecteurs de dimension 3 : `U = (x1, y1, z1)` et `V = (x2, y2, z2)`

La somme vaut `W = U+V = (x1+x2, y1+y2, z1+z2)`

- Le produit scalaire vaut `p = U.V = x1*x2 + y1*y2 + z1*z2`
- La norme vaut la racine carrée du produit scalaire du vecteur avec lui-même
- Pour normaliser un vecteur on divise toutes ses composantes par sa norme
- Le produit vectoriel de 2 vecteurs est un vecteur :

`Z= U×V = (y1*z2-z1*y2, z1*x2-x1*z2, x1*y2-y1*x2)`

Exercice 2 :

Écrire un programme permettant de saisir une liste d'un nombre quelconque d'entiers strictement positifs (le nombre est toutefois borné par 100). Une valeur négative permet d'arrêter la saisie. Le programme réaffiche d'abord les valeurs saisies par l'utilisateur, puis les mélange pour les afficher dans un ordre aléatoire.

```
Entre un entier positif : 1
Entre un entier positif : 2
Entre un entier positif : 3
Entre un entier positif : 4
Entre un entier positif : 5
Entre un entier positif : -1
Vous avez saisi :
1  2  3  4  5
Je mélange... :
3  4  1  5  2
```

Pour mélanger on procède comme ceci : soit *n* le nombre d'entiers saisis. On répète *n* fois la procédure :

- choisir au hasard deux indices compris entre *0* et *n-1*
- échanger le contenu des cases du tableau correspondant à ces deux indices.

Après *n* itérations, le tableau a de bonnes chances d'être mélangé.

On dispose d'une fonction `random()` qui retourne un réel aléatoire dans l'intervalle `[0;1.0[`

TD 7 – Corrections

Exercice 1 :

```
# affichage à l'envers
programme VecteurEnvers
début
  avec vecteur[3]: réel
       i         : entier

  pour i de 0 à 2 faire
    saisir vecteur[i]
  finfaire

  pour i de 2 à 0 pas -1 faire
    afficher vecteur[i]
  finfaire

fin VecteurEnvers
```

```
# norme d'un vecteur
programme VecteurNorme
début
  avec vec[3] : réel
       i      : entier
       norme  : réel

  pour i de 0 à 2 faire
    saisir vec[i]
  finfaire

  norme <- 0.0

  pour i de 0 à 2 faire
    norme<-norme+vec[i]*vec[i]
  finfaire

  norme <- sqrt(norme)

  afficher norme

  pour i de 0 à 2 faire
    vec[i] <- vec[i]/norme
    afficher vec[i]
  finfaire

fin VecteurNorme
```

```
# addition et produit scalaire
programme VecteursAdd
début
  avec vec1[3], vec2[3] : réel
       i                : entier
       scalaire         : réel

  pour i de 0 à 2 faire
    saisir vec1[i]
  finfaire

  pour i de 0 à 2 faire
    saisir vec2[i]
  finfaire

  afficher "La somme vaut :"

  pour i de 0 à 2 faire
    afficher vec1[i]+vec2[i],"  "
  finfaire

  scalaire <- 0.0

  pour i de 0 à 2 faire
    scalaire<-scalaire+vec1[i]*vec2[i]
  finfaire

  afficher scalaire
fin VecteurAdd
```

```
# produit vectoriel
# éviter le réflexe systématique de vouloir
# utiliser des boucles !
# il faut s'adapter au problème, et pas
# l'inverse !
programme ProdVec
début
  avec vec1[3], vec2[3]: réel
       i               : entier
       res[3]          : réel

  pour i de 0 à 2 faire
    saisir vec1[i]
  finfaire

  pour i de 0 à 2 faire
    saisir vec2[i]
  finfaire

  res[0] <- vec1[1]*vec2[2]-vec2[1]*vec1[2]
  res[1] <- vec2[0]*vec1[2]-vec1[0]*vec2[2]
  res[2] <- vec1[0]*vec2[1]-vec1[1]*vec2[0]

  pour i de 0 à 2 faire
    afficher res[i],"  "
  finfaire
fin ProdVec
```

Exercice 2 :

Sur la méthode, une proposition est la suivante : on écrit d'abord le programme principal tel qu'on aimerait qu'il fonctionne. Cela doit permettre de :

- faire les choix fonction / procédure
- bien analyser pour chacune la liste des paramètres le type de passage (valeur ou référence)

```
// Saisir et mélanger une liste d'entiers
programme Melange
début
  avec. li[100] : entier   # les nombres de la liste saisie
        n       : entier.  # nombre d'entiers dans la liste
        i       : entier   # compteur
  saisie_liste(;li, n)  # on choisit ici une procédure avec 2 paramètres de sortie
  afficher "vous avez saisi :"
  affiche_liste(li, n)
  afficher "je mélange… :"
  melange_liste(;li, n) # on considère le "package tableau/entier" donc on passe
                        # tout par référence, mais n aurait pu être passé par valeur.
fin Melange
```

On écrit maintenant les procédures. Avoir passé du temps à les concevoir (savoir précisément ce qu'elles doivent faire et de quels paramètres elles ont besoin) nous permet d'être efficaces ici.

```
#-------------------------------------------------------------------------------
# procedure qui effectue la saisie continuellement : la saisie s'arrête lorsqu'on
# entre un nb <0 ou lorsque l'on a entré 100 valeurs
# en entrée/sortie : la liste et l'entier associé
#-------------------------------------------------------------------------------
procedure saisie_liste( ; lst[100] : entier, nb : entier)
début
  nb <- 0
  afficher "Entre un entier positif : "
  saisir lst[nb]
  tant que lst[nb]>=0 et nb<100 faire
      nb <- nb + 1
       if nb<100
          saisir lst[nb]
  fin faire
fin

#-----------------------------------------------
#...
#-----------------------------------------------
procedure affiche_liste(liste[100] : entier, nombre : entier)
début

  pour i de 0 à nombre-1 faire
       afficher liste[i]
  finfaire

fin

#-----------------------------------------------
...
#-----------------------------------------------
procedure melange_liste( ; l[100] : entier, n : entier)
début
       avec   tampon           : entier   # pour l'échange des contenus
       avec   indice1, indice2 : entier   # les deux indices pour l'échange
  pour i de 0 à n-1 faire
       indice1 <- entier(random()*réel(n-1))
       indice2 <- entier(random()*réel(n-1))
       tampon <- l[indice1]
       l[indice1] <- l[indice2]
       l[indice2] <- tampon
  finfaire
fin
```

TP 7

Exercice 1, application directe du cours :

Écrivez des programmes permettant de tester les exemples en python de manipulation de tableaux vus en cours (remplissage d'un tableau de 10 valeurs et son affichage, saisie et affichage d'un tableau à deux dimensions, passage de paramètres).

Exercice 2 :

Écrire un programme (avec un menu) permettant de :

1. Saisir un nom et afficher le nombre de voyelles
2. Saisir un nom et vérifier qu'il s'agit d'un palindrome.
3. Quitter

Rappels :

- le type str est manipulable "comme" un tableau :

```
if __name__ == "__main__":
    chaine : str
    longueur : int
    chaine = input("entrez une chaine : ")
    print("le premier caractère est",chaine[0])
    longueur = len(chaine)
    print("la chaine fait",longueur,"caractere(s)")
    print("le dernier caractère est",chaine[longueur-1])
```

- un palindrome est un texte ou un mot dont l'ordre des lettres reste le même qu'on le lise de gauche à droite ou de droite à gauche : « kayak », « elle ». Nous considèrerons également que majuscules et minuscules sont des lettres différentes (« Laval » ne sera pas un palindrome, « LAVAL » en sera un)

Exercice 3 :

On veut effectuer les mêmes opérations que celles de l'exercice 2 sur des noms stockés dans un tableau (et non plus saisis un par un par l'utilisateur).
La saisie des noms, successivement placés dans un tableau, s'effectue tant que l'utilisateur répond 'o' à la question "Encore un nom ?"

Le menu devient donc :

```
1- Saisir un tableau de noms
2- afficher le nombre de voyelles par nom
3- afficher uniquement les noms étant des palindromes
4- quitter
```