# TP01 : Montée en compétences Lisp

**Livrables attendus : un fichier lisp + un rapport présentant et argumentant le code lisp proposé.**

**Date de remise : 04 octobre 2020 à 18H**

## Exercice 1 : Mise en condition (8 points)

Définir les fonctions suivantes :

reverseA (arg1 arg2 arg3) : retourne la liste constituée des trois arguments dans l'ordre inverse.

*(reverseA 1 2 3) -> (3 2 1)*

reverseB (L) : retourne la liste inversée de 1, 2 ou 3 éléments.

*(reverseB '(1 2) ) -> (2 1)*

reverseC (L ) : retourne la liste constituée des éléments de L inversés.

*(reverseC '( a b (c d) e f)) -> (f e (c d) b a)*

double (L) : retourne la liste constituée des éléments de L en doublant les atomes.

*(double '((1 2) 3 (4 5) 6)) -> ((1 2) 3 3 (4 5) 6 6)*

nombres3 ( L) : retourne BRAVO si les 3 premiers éléments de la liste L sont des nombres, sinon PERDU.

*(nombres3 '( 1 2 3 R S 4) -> BRAVO*

grouper (L1 L2) : retourne la liste composée des éléments successifs des deux listes passées en arguments L1 et L2.

*(grouper '(1 2 3) '(4 5 6)) -> ((1 4)(2 5)(3 6))*

monReverse(L) : retourne la liste inversée de L

*(monReverse '(1 2 3 4 5)) -> (5 4 3 2 1)*

palindrome(L) : retourne vrai si L est un palindrome

*(palindrome '(x a m a x) ) -> T*

## Exercice 2 : Objets fonctionnels (1 point)

L'objectif est de voir comment utiliser mapcar avec des fonctions anonymes, des expressions *lambda*. Une fonction anonyme est une fonction sans nom qui peut être fournie en paramètre à toute fonction LISP. Le format est:

```
(lambda (paramètres)
  body)
```

où <body> est une suite d'instructions LISP. Par exemple:

```
(lambda (x)
  (* x 3))
```

est une fonction anonyme qui retourne le triple de l'argument passé en paramètre.

Ecrivez une fonction *list-triple-couple* qui retourne la liste des couples composés des éléments de la liste fournie en paramètre et de leur triple.

Utilisez mapcar.

Exemple:

(list-triple-couple '(0 2 3 11)) -> ((0 0)(2 6)(3 9) (11 33))

# Exercice 3 : *a-list (1 point)*

Ecrivez les trois fonctions décrites ci-dessous.

*Arguments :*
clé : une S-Expression quelconque ;

a-liste : une liste d'association, c'est-à-dire une liste de la forme :

($(clé_1\ valeur_1)\ (clé_2\ valeur_2)\ ...\ (clé_n\ valeur_n)$)

- my-assoc (cle a-list) : retourne la valeur d'une clé dans une liste d'associations (A-liste)
- cles (a-list) : retourne la liste des clés d'une A-liste
- creation (listeCles listeValeurs) retourne une A-liste à partir d'une liste de clés et d'une liste de valeurs

*Spécification :*
(my-assoc cle a-liste) retourne nil si cle ne correspond à aucune clé de la liste d'association, la paire correspondante dans le cas contraire; cette fonction existe sous le nom de assoc.

*Exemples :*

```
> (my-assoc 'Pierre
     '((Yolande  25) (Pierre  22) (Julie  45)))
(Pierre  22)

> (my-assoc 'Yves
```

```
    '((Yolande  25) (Pierre  22) (Julie  45)))
nil

> (cles '((Yolande  25) (Pierre  22) (Julie  45)))
(Yolande Pierre Julie)

>(creation '(Yolande Pierre Julie) '(25 22 45))

((Yolande  25) (Pierre  22) (Julie  45))
```

# *Exercice 4 : gestion d'une base de connaissances en Lisp (10 points)*

Un cimetière comporte des tombes abritant des restes mortuaires. Une tombe sera caractérisée par le nom de son résident, l'année de l'inhumation de ce dernier, son emplacement, l'année du début de la concession (location de la parcelle de terrain) et la durée de celle-ci.

On supposera que les tombes sont disposées en rangées parallèles. Ainsi la position d'une sépulture sera déterminée par un couple de nombres~: le numéro de la rangée et le numéro de la tombe dans la rangée.

Les informations relatives à une tombe seront stockées sous forme d'une liste : par exemple `("Bécaud" 2001 (45 17) 2000 30)` représente la tombe de Gilbert Bécaud, inhumé en 2001 dans la 17e tombe de la 45e rangée et dont la concession a débuté en 2000 pour une durée de 30 ans.

Un cimetière est représenté par une liste de tombes. Les cimetières sont à leur tour stockés dans une liste de *cimetieres*. Exemple de cimetière :

```
(pere-lachaise (
   ("Bécaud" 2001 (45 17) 2000 30)   ("Desproges" 1988 (11 6) 1988 30)
   ("Grappelli" 1997 (85 23) 1997 5) ("Morrison" 1971 (6 12) 1971 30)
   ("Mouloudji" 1994 (42 9) 1990 15) ("Nohain" 1981 (89 14) 1979 15)
   ("Oussekine" 1986 (85 37) 1986 5) ("Petrucciani" 1999 (11 26) 1999 15)
   ("Popesco" 1993 (85 16) 1985 30)  ("Signoret" 1985 (44 7) 1980 30
   ("Zavatta" 1993 (11 16) 1993 15))
```

**NB :** les exemples donnés ne servent qu'à des fins d'illustration ; vous devez écrire des fonctions valables pour des tombes et des cimetières quelconques. L'élégance des solutions proposées entre en compte dans la notation.

### *Question 1 (3 points)*

Écrire les différentes fonctions d'accès aux différentes composantes d'une tombe : `nom`, `an-inhum`, num, rangee, `debut-loc`, `durée-loc`.

### *Question 2 (1 point)*

Écrire une fonction `qui-est-là` qui, étant donné un emplacement (numéro + rangée) et un cimetière, retourne le nom de la personne qui y est enterrée.

Exemple :

```
> (qui-est-la '(11 16) ‘pere-lachaise ‘cimetieres)
"Zavatta"
> (qui-est-la '(73 11) ‘pere-lachaise)
"Emplacement non attribué"
```

### *Question 3 (1 point)*

Spécifier puis écrire un prédicat `prévoyant?` qui, étant donné une tombe, retourne vrai si son occupant a acheté sa concession par anticipation (avant son décès), faux sinon. Pour cette question ne pas utiliser de structure de contrôle conditionnelle.

### *Question 4 (1 point)*

Écrire une fonction `nb-prévoyants` qui, étant donné un cimetière, compte le nombre de résidents ayant acheté leur concession par anticipation.

Exemple :

```
> (nb-prevoyants ‘pere-lachaise ‘cimetieres)
5
```

### *Question 5 (2 points)*

Écrire une fonction annuaire qui, étant donné un cimetière et un entier naturel n, retourne la liste des noms des résidents de la rangée n.

Exemple (ne pas tenir compte de l’ordre des réponses) :

```
> (annuaire ‘pere-lachaise 85 ‘cimetieres)
("Grappelli" "Oussekine" "Popesco")
```

### *Question 6 (2 points)*

Écrire une fonction doyen-benjamin qui, étant donné un cimetière, retourne le résident le plus ancien et le résident le plus récent. On s’attachera à proposer une solution à la fois la plus élégante et la plus efficace possible.

Exemple :

```
> (doyen-benjamin ‘pere-lachaise ‘cimetieres)
(("Morrison" 1971 (6 12) 1971 30) ("Bécaud" 2001 (45 17) 2000 30))
```