

# TP01 : Montée en compétences Lisp

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

**Date de remise : 23 octobre 2022 à 18H**

## Exercice 1 : Mise en condition

1. Déterminez le type des objets lisp suivant :

35 **entier**

(35) **liste**

((3) 5) 6) **liste imbriquée**

-34RRRR

T **Booléen (vrai)**

NIL **Booléen (faux)**

() **liste vide**

2. Traduisez sous forme d'arbre la liste suivante :

((A)(B)(C)) G (((D)) F) H))

Quel est l'objet le plus profond ?

3. Que font les appels de fonctions suivants :

(CADR (CDR (CDR (CDR '(DO RE MI FA SOL LA SI))))) **SOL**

(CONS (CADR '((A B)(C D))) (CDDR '(A (B (C))))) **((C D))**

(CONS (CONS 'HELLO NIL) '(HOW ARE YOU)) **((HELLO) HOW ARE YOU)**

(CONS 'JE (CONS 'JE (CONS 'JE (CONS 'BALBUTIE NIL)))) **(JE JE JE BALBUTIE)**

(CADR (CONS 'TIENS (CONS '(C EST SIMPLE) ()))) **(C EST SIMPLE)**

4. Ecrire les fonctions suivantes :

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

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*

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<sub>1</sub> valeur<sub>1</sub>) (clé<sub>2</sub> valeur<sub>2</sub>) ... (clé<sub>n</sub> valeur<sub>n</sub>))

- `my-assoc` (`cle a-list`) : 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`.
- `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

*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***

On considère une base de connaissances sur les guerres de France<sup>1</sup> définies par les propriétés suivantes : nom du conflit, date de début de conflit, date de fin de conflit, belligérants (alliés y compris la France, ennemis), lieu) .

Soit la base `BaseTest` suivante comprenant un seul conflit :

```
(setq BaseTest
```

```
' ( ("Guerre de Bourgondie" 523 533 (("Royaume Franc") ("Royaume des Burgondes")) ("Vezeronce"
"Arles"))))
```

**A. Compléter `BaseTest` avec les conflits commençant avant l'an 1100<sup>2</sup>.**

**B. Définir les fonctions de service :**

**`dateDebut (conflit)` :** retourne la date de début du conflit passé en argument

```
> (dateDebut '("Guerre de Bourgondie" 523 533 (("Royaume Franc") ("Royaume des Burgondes"))
("Vezeronce" "Arles")))
```

```
523
```

**`nomConflit (conflit)` :** retourne le nom du conflit passé en argument

```
> (nomConflit '("Guerre de Bourgondie" 523 533 (("Royaume Franc") ("Royaume des Burgondes"))
("Vezeronce" "Arles")))
```

```
" Guerre de Bourgondie "
```

---

<sup>1</sup> [https://fr.wikipedia.org/wiki/Liste\\_des\\_guerres\\_de\\_la\\_France](https://fr.wikipedia.org/wiki/Liste_des_guerres_de_la_France)

<sup>2</sup> Vous référer au site wikipedia ; [https://fr.wikipedia.org/wiki/Liste\\_des\\_guerres\\_de\\_la\\_France](https://fr.wikipedia.org/wiki/Liste_des_guerres_de_la_France)

**allies (conflit)** : retourne les alliés de du conflit passé en argument

```
>(allies '("Guerre de Bourgondie" 523 533 (("Royaume Franc") ("Royaume des Burgondes"))  
("Vezeronce" "Arles")))
```

```
("Royaume Franc")
```

**ennemis (conflit)** : retourne les ennemis du conflit passé en argument

```
>(ennemis '("Guerre de Bourgondie" 523 533 (("Royaume Franc") ("Royaume des Burgondes"))  
("Vezeronce" "Arles")))
```

```
("Royaume des Burgondes")
```

**lieu (conflit)** : retourne le lieu du conflit passé en argument

```
>(lieu '("Guerre de Bourgondie" 523 533 (("Royaume Franc") ("Royaume des Burgondes"))  
("Vezeronce" "Arles")))
```

```
("Vezeronce" "Arles")
```

**C. En utilisant ces fonctions de service si nécessaire, définir les fonctions suivantes :**

FB1 : affiche tous les conflits

FB2 : affiche les conflits du "Royaume Franc"

FB3 : retourne la liste des conflits dont un allié est précisé en argument

FB4 : retourne le conflit dont la date de début est 523

FB5 : retourne la liste des conflits dont la date de début est comprise entre 523 et 715

FB6 : calcule et retourne le nombre de conflits ayant pour ennemis les "Lombards"