

Cours IA ESP-UCAD

Pr. Mamadou Camara
mamadou.camara@esp.sn

2021-2022

Table des matières

1	Introduction au Lisp	2
1.1	TD les premières fonctions	2
1.1.1	Exercice : Quote [DIC-DIT Info]	2
1.1.2	Exercice : Cons, car, cdr et Quote [Master Gl]	2
1.1.3	Exercice : atom	3
1.2	TD Block	3
1.2.1	Exercice : Définition implicite : block nil	3
1.2.2	Exercice : Définition implicite, block fonction [DIC Projet 2020]	3
1.3	TD les fonctions à définir [DIC Projet 2020]	4
1.3.1	Exercice : produit scalaire	4
1.3.2	Exercice : Nombre d'atomes	4
1.3.3	Exercice : Remplacement	4
1.4	TD les fonctions à définir [Master Gl]	5
1.4.1	Exercice : position (5 points)	6
1.4.2	Exercice : Append	7
1.4.3	Exercice : Moyenne	7
1.4.4	Exercice : Un exemple de filtrage	7
1.5	TD les fonctions à définir [DIT Info]	7
1.5.1	Exercice : Puissance	7
1.5.2	Exercice : Nombre d'apparitions	8
2	Annexe	9
2.1	Annexe1	9
2.1.1	Exécution de fichier	9
2.1.2	Commentaires	9
2.1.3	Effacer écran	9
2.2	Lisp	10

Chapitre 1

Introduction au Lisp

1.1 TD les premières fonctions

1.1.1 Exercice : Quote [DIC-DIT Info]

Donner la sortie de chacune des lignes suivantes :

1. 'Jean
2. '(Jean D Publique)
3. '2
4. 2
5. '(+ 4 3)
6. (+ 4 3)
7. (list 1 2 (+ 3 4) '(* 5 6))
8. Jean
9. (Jean D Publique)
10. (append '(Paul Karim) (list '(Jean D Publique) 'Sandra))
11. (length (append '(Paul Karim) (list '(Jean D Publique) 'Sandra)))
12. (or (atom '()) (numberp "Salut"))

1.1.2 Exercice : Cons, car, cdr et Quote [Master G1]

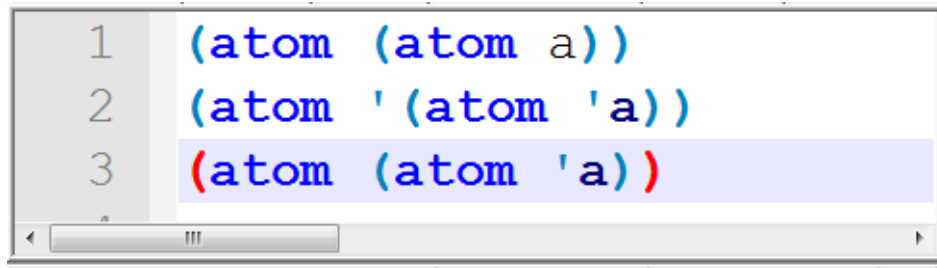
Donner la sortie de chacune des lignes suivantes :

1. a
2. 'a
3. (1 2 3)
4. (cons '(+ 3 4) '(2 3))
5. (cadr '(((+ 3 4) (2) ((3)))))
6. (cddr '(((+ 3 4) (2) ((3)))))
7. (cons 'Salut 'camarade)
8. (cons 'Salut camarade)
9. (cons 'Salut camarade '(((4) 5))
10. (cons 'Salut 'camarade '(((4) 5))
11. (cons "Salut camarade!" '(((4) 5))
12. (cons (+ 6 7) '(4 5))

13. `(cons '(+ 6 7) '(4 5))`
14. Reformuler l'appel suivant sur la variable `x` en utilisant une combinaison de `car` et de `cdr` ¹
— `(cddadr x)`

1.1.3 Exercice : atom

Donner la sortie des instructions suivantes et justifier.

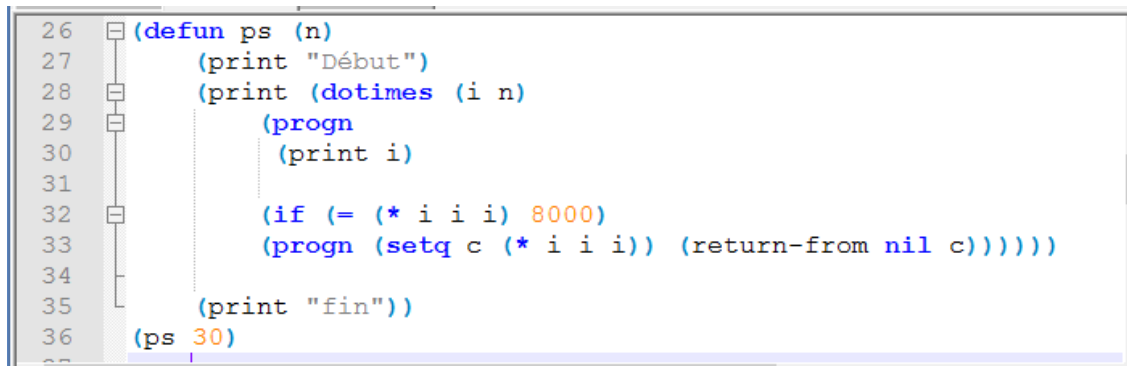


```
1 (atom (atom a))
2 (atom '(atom 'a))
3 (atom (atom 'a))
```

1.2 TD Block

1.2.1 Exercice : Définition implicite : block nil

Donner la sortie des instructions suivantes et justifier.



```
26 (defun ps (n)
27   (print "Début")
28   (print (dotimes (i n)
29     (progn
30       (print i)
31
32       (if (= (* i i i) 8000)
33         (progn (setq c (* i i i)) (return-from nil c))))))
34
35   (print "fin"))
36 (ps 30)
```

1.2.2 Exercice : Définition implicite, block fonction[DIC Projet 2020]

Donner la sortie des instructions suivantes et justifier.

-
1. Nous supposons que `x` a déjà une valeur

```

1  (defun consec (X Y L)
2    (print "bonjour")
3    (print (length L))
4    (cond
5      ((null L) nil)
6      ((null (cdr L)) nil)
7      ((and (eq (car L) X) (eq (cadr L) Y))
8        (return-from consec (print
9          (list X "et" Y "sont Consécutif dans la liste" L))))
10     (t (consec X Y (cdr L))))
11  )
12  (print (length L))
13  (print "Au revoir")
14  )
15
16  (consec 'a 'b '(c f g a b g))
17
18  (consec 'g 'b '(c f g a b g))
19

```

1.3 TD les fonctions à définir [DIC Projet 2020]

1.3.1 Exercice : produit scalaire

Écrire la fonction qui calcule le produit scalaire de deux vecteurs. Proposer les versions qui utilisent :

1. la fonction "let" avec une variable, un compteur et la fonction dotimes.
2. une combinaison des fonctions "apply" et "mapcar".
3. la récursivité

Les valeurs excédentaires dans l'une ou l'autre des listes ne sont pas prise en compte.

1.3.2 Exercice : Nombre d'atomes

Écrire une fonction qui compte le nombre d'atome dans une liste. Il s'agit de compter les atomes différents de nil à tous les niveaux (profondeurs) de la liste. L'appel de la fonction sur un atome doit renvoyer une erreur.

Appel	Résultat
(decompte '(k p (a b c d) (e) f))	8
(decompte '((u v w) x y))	5
(decompte '((a b c d) (e) f))	6
(decompte 'a)	erreur

1.3.3 Exercice : Remplacement

Écrire une fonction Remplacer qui prend comme arguments deux atomes et une liste et qui remplace toutes les occurrences du premier atome dans la liste par le second. Écrire une fonction substitution qui prend une liste L et autre liste S constituée de paires (liste a deux éléments atomiques) et utiliser la fonction remplacer pour faire des substitutions dans la première liste selon les correspondances établies dans la seconde liste. Dans le cas où un des éléments de la seconde liste n'est pas une paire la fonction substitution doit avancer sans rien faire. Le remplacement se fait au premier niveau dans L, c'est-à-dire sur les éléments de L et non sur les éléments des éléments de L.

```

36 (print "Sortie ligne 37")
37 (print (remplacer 't 'ti '(d t k 2 y j k i 10)))
38 (print "Sortie ligne 39")
39 (print (substituer '(d t k 2 y j k i 10) '((t ti) c (g h u) (10 dix) ((y)igrec) )))
40 (print "Sortie ligne 41")
41 (print (substituer '(d t k 2 y j k i 10) 'e))

```

```

GNU CLISP 2.49
[91] <load "C:/Code/Lisp/Material/lp/sub.lisp">
;; Loading file C:/Code/Lisp/Material/lp/sub.lisp ...
"Sortie ligne 37"
<D T I K 2 Y J K I 10>
"Sortie ligne 39"
<D T I K 2 Y J K I DIX>
"Sortie ligne 41"
<D T K 2 Y J K I 10>
;; Loaded file C:/Code/Lisp/Material/lp/sub.lisp
T
[10] > █

```

Pour ce qui est de S, il faut utiliser seulement ses éléments qui sont des paires d'atomes. Supposons les conditions suivantes :

1. La liste L est nulle
2. La liste S est nulle
3. S est atomique
4. Le premier élément de S est atomique
5. Le premier élément de S est une liste avec 1 seul élément
6. Le premier élément du premier élément de S n'est pas atomique
7. Le second élément du premier élément de S n'est pas atomique
8. Le premier élément de S contient exactement deux éléments atomiques
9. Autres situations

1.4 TD les fonctions à définir [Master GI]

1.4.1 Exercice : position (5 points)

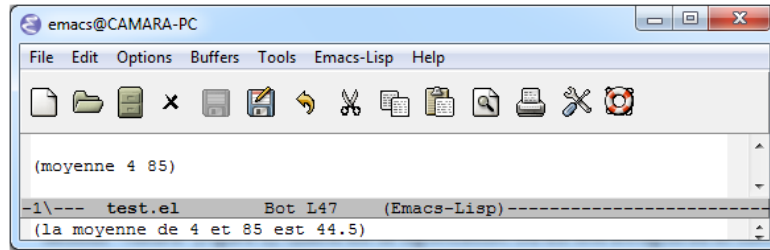
Écrire une fonction récursive qui renvoie la position à laquelle un élément e apparaît la première fois dans une liste L (traitement au premier niveau de la liste). La fonction doit renvoyer nil si l'élément est absent de la liste.

1.4.2 Exercice : Append

Redéfinir de façon récursive la fonction `append` qui prend en argument une liste `l` et une liste `m` et va renvoyer leur concaténation. L'ordre des éléments dans la liste ainsi construite n'est pas importante. Il n'est nécessaire de définir aucune variable dans la fonction.

1.4.3 Exercice : Moyenne

Écrire en Lisp la fonction `moyenne` qui prend en argument un élément `x` et un élément `y` pour renvoyer une liste dont la structure est décrite dans la figure suivante. La fonction devra utiliser une expressions "let" utilisant une ou deux variables.



1.4.4 Exercice : Un exemple de filtrage

Deux listes sont équivalentes à & près, si elles donnent la même chose lorsqu'on leur retire les & Ainsi

```
33 (print (eqv '(A & & B C D &) '(& & A B C D &)))
34 ; => t
35 (print (eqv '(A & & B C D &) '(& & A B C E &)))
36 ; => nil
37 (print (eqv '(& A & & B & C D &) '(& & A B C D & &)))
38 ; => t
39 (print (eqv '(& A & & & C D &) '(& & A B C D & &)))
40 ; => nil
41 (print (eqv '(& A & & G & C D &) '(& & A B C D & &)))
42 ; => nil
```

Créer ce nouveau prédicat, on va tester l'égalité des deux liste des deux listes en utilisant les fonctions `f1` et `f2`.

1. `f1` : Renvoie le premier élément différent de &
2. `f2` : Renvoie la liste commençant par premier lettre différent de &

1.5 TD les fonctions à définir [DIT Info]

1.5.1 Exercice : Puissance

Écrire la fonction `puissance` de façon récursive.

Appel	Résultat
(puissance 2 3)	8
(puissance 2 (- 3))	0.125
(puissance 5 (- 9))	5.12 e-007

Il est demandé d'écrire une seconde fonction qui permettra à l'utilisateur de donner saisir les entrées, d'exécuter la fonction puissance et d'afficher le résultat.

1.5.2 Exercice : Nombre d'apparitions

Écrire le nombre de fois qu'une expression apparaît n'importe où (décompte à tous les niveaux) dans une autre expression.

Appel	Résultat
(occurrence '(a b c) '(a b c))	0
(occurrence 'a '(a d c a b))	2
(occurrence '(a) '(d (a) a))	1
(occurrence '(a) '(d a h a))	0
(occurrence '((a c)) '(d ((a c)) h (d r) (a c) a v (a c)))	1
(occurrence '(a c) '(d ((a c)) h (d r) (a c) a v (a c)))	3
(occurrence '(a b c) '((a b c d) a b c))	0

Chapitre 2

Annexe

2.1 Annexe1

2.1.1 Exécution de fichier

— (load "C :/Lecture/M1/IA/Lisp/Material/tp2/testertp2.lisp")

2.1.2 Commentaires

1. commentaire sur une ligne ;;
2. commentaires sur plusieurs lignes commencent par `#|` et se terminent par `|#`

2.1.3 Effacer écran

1. (shell)
2. cls
3. exit

2.2 Lisp

Donner les sorties des instructions suivantes :

1. `(cadr '((+ 3 4) (2) ((3))))`
2. `(caddr '((+ 3 4) (2) ((3))))`
3. `(cons 'Salut 'camarade)`
4. `(cons 'Salut camarade)`
5. `(cons 'Salut 'camarade '((4) 5))`
6. `(cons "Salut camarade!" '((4) 5))`
7. `(cons (+ 6 7) '(4 5))`
8. `(cons '(+ 6 7) '(4 5))`