



# **Cours 2 - Programmation Lisp (I)**

MARIE-HÉLÈNE ABEL

# Table des matières

<b>I - Valeurs logiques et prédicats</b>	<b>5</b>
A. Opérateurs logiques.....	5
B. Les prédicats.....	6
C. Les semi-prédicats.....	6
<b>II - Unité de base d'un programme lisp</b>	<b>7</b>
A. La forme spéciale defun.....	7
B. La fonction anonyme.....	8
<b>III - Opérateur conditionnel</b>	<b>9</b>
<b>IV - Approches de programmation</b>	<b>11</b>

# Valeurs logiques et prédicats

Opérateurs logiques	5
Les prédicats	6
Les semi-prédicats	6

Lisp peut exprimer comme la plupart des langages les valeurs logiques représentant vrai et faux.

- Vrai est représenté par le symbole T
- Faux est représenté par le symbole NIL

Ces deux valeurs s'auto-évaluent et ne peuvent être redéfinies.

NIL représente également la liste vide.

## A. Opérateurs logiques

Un certain nombre d'opérateurs logiques sont prédéfinis (primitives) : **OR**, **AND**, **NOT**



### Exemple

```
> (or nil nil nil)
NIL
> (or nil t nil)
T
> (and t nil t)
NIL
> (and t)
T
> (not nil)
T
> (setq flag (or nil nil t))
T
>
```

## B. Les prédicats



### Définition

Les prédicats sont des fonctions qui retournent des valeurs logiques (T ou NIL) : **NUMBERP**, **LISTP**, **FLOATP**, **INTEGERP**, **NULL**, **SYMBOLP**, **STRINGP**



### Exemple

```
>(numberp 34)
T
>(listp '(or nil nil t))
T
>(floatp 2.31)
T
>(integerp 2.31)
NIL
>(null nil)
T
>(symbolp "Albert")
NIL
>(stringp 'Albert)
NIL
>
```

## C. Les semi-prédicats



### Définition

Un semi-prédicat est une fonction qui renvoie faux (nil), ou une autre valeur quelconque, différente de nil, qui sera interprétée comme vraie.

La primitive **member**, les opérateurs logiques **and**, **or** sont des semi-prédicats.



### Exemple

```
>(or (member 4 '(1 2 3 4 5 'Albert)) t nil t nil)
(4 5 'ALBERT)
>
```

# Unité de base d'un programme lisp

La forme spéciale defun

7

La fonction anonyme

8

Pour être utile, un langage doit laisser l'utilisateur définir ses propres fonctions, qui lui serviront ensuite d'éléments de bibliothèque.

## A. La forme spéciale defun



### Définition

**defun** (pour "define function") permet de définir une fonction utilisateur nommée.



### Exemple : Définition de la fonction square

```
> (defun square (xx) (* xx xx))
SQUARE
>
```



### Syntaxe : Syntaxe simplifiée

```
(defun <nom de la fonction> <liste des arguments>
  <commentaire optionnel>
  <expr1>
  <expr2>
  ...
  <exprN>)
```



### Remarque

Le résultat de la fonction est toujours la dernière évaluation (<exprN>).  
Une fois qu'une fonction est définie, elle peut être appliquée comme n'importe quelle autre primitive.



### Exemple : Application de la fonction square

```
> (square 121)
14641
> (square "Julius")
> Error: Argument "Julius" is not of type NUMBER.
```

```
> While executing: CCL::TOPLEVEL-EVAL  
> Type Command-. to abort. See the Restarts... menu item for further choices.  
1 >
```

## B. La fonction anonyme

### Préambule

Il n'est pas nécessaire lorsque l'on veut définir une fonction de lui donner un nom. On peut utiliser une notation particulière appelée **lambda-notation** pour exprimer une fonction qui n'aura pas d'identité.



### Exemple : Construction d'une liste à deux éléments identiques

```
> ((lambda (xx) (list xx xx)) 'a)  
(A A)  
>
```



### Remarque

Les fonctions anonymes servent surtout pour définir des fonctions de service utilisées temporairement à l'intérieur d'autres fonctions (cf. mapping).



### Attention : Une lambda-expression n'est pas une forme Lisp standard

- Elle a un statut spécial et est considérée équivalente à un nom de fonction.
- On ne peut évaluer une expression commençant par lambda.

# Opérateur conditionnel



## Définition

Un opérateur conditionnel permet d'exécuter du code sous condition.  
L'opérateur conditionnel Lisp le plus simple est : **if**



## Syntaxe

(if forme-test  
Forme1  
Forme2)



## Méthode : Une expression (if ...) a sa propre règle d'évaluation :

- La première expression est évaluée (forme-test),
- Si la valeur de test est vrai alors Forme1 est évaluée et sa valeur retournée
- Sinon Forme2 est évaluée et sa valeur retournée.



## Exemple

```
>(setq a 2)  
2  
>(if (= a 2) t nil)  
T  
>
```

# Approches de programmation

## IV



### Définition : Approche ascendante (ou bottom-up)

Part de la structure de représentation des données pour construire des fonctions élémentaires avant de passer à des choses plus compliquées.



### Définition : Approche descendante (ou top-down)

Part du problème à résoudre et détaille progressivement les fonctions à réaliser pour terminer ensuite par les structures de données.



### Remarque

En Lisp on utilise souvent une combinaison des deux approches.