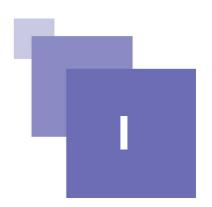
# Cours 8 Programmation LISP (IV)

Marie-Hélène Abel

# Table des matières

I - Le	s fonctions d'entrée-sortie	5
	A. Préambule	.5
	B. Lecture	.5
	C. Ecriture	6
II - L	es fichiers	9
	A. Chargement d'un fichier	.9
	B. Ouverture d'un fichier	. <b>9</b> 9
	2. La macro with-open-file	
	D. Destruction d'un fichier	11

# Les fonctions d'entrée-sortie



Préambule	5
Lecture	5
Ecriture	6

#### A. Préambule

De nombreuses macros ou fonctions permettent de lire (read) et écrire (print) sur des flots ouverts.



#### Méthode : Les fonctions d'écriture

- Les fonctions d'écriture telles que print écrivent par défaut un objet sur la sortie standard (flot prédéfini et toujours ouvert, référencé par la variable globale \*standard-output\*).
- Un argument mot-clé permet d'indiquer un autre flot.



#### Méthode : Les fonctions de lecture

Les fonctions de lecture lisent par défaut sur l'entrée standard \*standard-input\* et acceptent un argument mot-clé qui change le comportement par défaut.

#### **B.** Lecture



#### Définition : La fonction read

La fonction read lit une expression lisp et la retourne.



#### Exemple

>(read)

Pomme

Pomme

>



#### Attention

La fonction **read** attend indéfiniment qu'une expression lisp complète soit tapée pour ensuite la retourner.

#### C. Ecriture

- Il existe plusieurs formats d'écriture : print, princ.
- La fonction **format** est la fonction générale d'écriture qui permet de formater du texte



#### Syntaxe: La fonction format

(format destination contrôle param-1 param-2 ...)



#### Méthode

- 1er argument obligatoire : destination. Il spécifie où écrire (fenêtre, fichier ...)
- 2ème argument :chaîne de caractères de contrôle pour spécifier quoi écrire et comment écrire les arguments optionnels (param-1, param-2 ...)

#### Différentes valeurs de destination

- T : écrire dans la fenêtre de l'interacteur
- Nil : écrire dans une chaîne de caractères qui est retournée comme valeur de l'appel format



#### Exemple

>(format t " Introduire un nombre : ")

Introduire un nombre :

Nil

>(format nil " Introduire un nombre : ")

"Introduire un nombre :"

>

#### Différentes directives à insérer dans la chaîne de contrôle

- ~%: impose un saut à la ligne.
- ~& : impose un saut de ligne, sauf si le curseur est déjà en début de ligne.
- ~a indique une position à remplir par un des arguments optionnels dans l'ordre d'apparition.
- ~{str~}: représente une construction itérative.



#### Remarque : ~{str~}

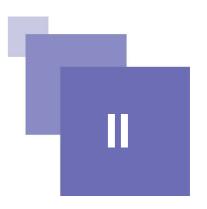
- L'argument doit être une liste, laquelle est utilisée comme un ensemble d'arguments comme pour un appel récursif de format.
- La chaîne str est utilisée comme contrôle.
- Chaque itération absorbe autant d'éléments de la liste qu'il est besoin d'arguments.



#### Exemple

```
>(format nil "~a plus ~a égale ~a" 2 3 5)
"2 plus 3 égale 5"
>(format nil " The winners are : ~{ ~a ~}." `(fred harry jill))
"The winners are : fred harry jill."
>
```

# Les fichiers



Chargement d'un fichier	9
Ouverture d'un fichier	9
Fermeture d'un fichier	11
Destruction d'un fichier	11

## A. Chargement d'un fichier



Syntaxe: La fonction Load

Load filename &key :verbose :print :if-does-not-exist



#### Méthode

Charge le fichier de nom filename dans un environnement lisp.

#### B. Ouverture d'un fichier

L'ouverture d'un fichier peut se faire à l'aide de la fonction **open** ou bien de la macro **with-open-file** 

## 1. La fonction Open



#### Syntaxe: La fonction Open

Open filename &key:direction:element type:if-exist:if-does-not-exist:external-format



#### Méthode

La fonction open permet d'ouvrir un flot sur un fichier.

- Le premier paramètre obligatoire correspond au chemin d'accès au fichier.
- Viennent ensuite des paramètres mots-clés, en particulier :direction qui indique le sens du flot, entrée ou sortie (en Anglais : input ou output).



#### Syntaxe::direction

Cet argument indique si le flot doit être en **input**, **ouput** ou les deux

- input, valeur par défaut,
- · :output

· :io



#### Syntaxe::element-type

Cet argument précise l'unité de transaction sur le flot

- **String-char**, unité de transaction par défaut. Les fonctions read-char et/ou write-char peuvent être utilisée sur le flot.
- Character, les fonctions read-char et/ou write-char peuvent être utilisée sur le flot.
- Signed-byte,
- Bit,
- etc.



#### Syntaxe::if-exists

Cet argument spécifie l'action à effectuer lorsque la direction est **output** ou **io** et que le fichier spécifié existe déjà.

- :error, action par défaut
- :new-version, crée une nouvelle version du fichier avec un numéro de version plus élevé
- **:rename**, renomme le fichier existant et crée le nouveau fichier avec le nom précisé.
- :rename-and-delete
- **:overwrite**, utilise le fichier existant. Les opérations d'écriture vont modifier le fichier. Si la direction est **io**, le pointeur de fichier est positionné en début de fichier. Ce mode est plus utilisé quand la fonction file-position peut être utilisée sur le flot.
- **:append**, utilise le fichier existant, les opérations d'écriture vont modifier le fichier. Le pointeur de fichier est positionné à la fin du fichier.



#### Syntaxe::if-does-not-exist

Cet argument spécifie l'action à effectuer si le fichier spécifié n'existe pas.

- :error
- :create
- nil, le flot est initialisé à nil.

### 2. La macro with-open-file



#### Définition : La macro with-open-stream

- La macro with-open-stream exécute un certain nombre d'opérations sur un flot ouvert, renvoie une valeur puis ferme le flot.
- Elle a deux paramètres. Le premier est un nom de variable qui prendra comme valeur le flot correspondant au deuxième paramètre.
- Le corps est une suite d'expressions formant un **progn** implicite.



#### Définition : La macro with-open-file

La macro with-open-file est similaire à la macro with-open-stream mais au préalable elle ouvre un flot sur un fichier avec open.



#### Exemple

>(defparameter \*flot-sortie\* (open "fichier" :direction :output))

```
*FLOT-SORTIE*
> (print '(1 2 3) *flot-sortie*)
(1 2 3)
> (print '(A B) *flot-sortie*)
(A B)
> (close *flot-sortie*)
T
> (with-open-file (flot-entree (open "fichier" :direction :input))
    (defparameter *expr1* (read flot-entree))
    (defparameter *expr2* (read flot-entree)))
*EXPR2*
> *expr1*
(1 2 3)
> *expr2*
(A B)
```

#### C. Fermeture d'un fichier



Définition : La fonction close

La fonction close ferme un flot ouvert.



#### Remarque

La connexion entre le flot et le fichier associé est terminée mais l'objet flot existe toujours en tant que flot non ouvert.

#### D. Destruction d'un fichier



#### **Syntaxe**

(delete-file file)



#### Remarque

L'argument file doit être une chaîne, un chemin ou un flot.