Tableaux

```
* (make-array '(4 3))

#2A((0 0 0) (0 0 0) (0 0 0) (0 0 0))
* (make-array '(4 3) :initial-element 5)

#2A((5 5 5) (5 5 5) (5 5 5) (5 5 5))
* (make-array 4 :initial-contents (list (+ 3 4) "hello" 'hi t))

#(7 "hello" HI T)
*
```

Tableaux

```
* (setf *x* (make-array 6 :fill-pointer 4))

#(0 0 0 0)
* (length *x*)

4
* (setf (fill-pointer *x*) 6)

6
* (length *x*)

6
* (adjust-array *x* '(30) :fill-pointer 20)

#(0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
*
```

Structures

```
* (defstruct person name address phone-number)

PERSON
* (make-person)

#S(PERSON :NAME NIL :ADDRESS NIL :PHONE-NUMBER NIL)
* (make-person :name "Bobby Beach")

#S(PERSON :NAME "Bobby Beach" :ADDRESS NIL :PHONE-NUMBER NIL)
* (defstruct person (name "unknown") address (phone-number 0))

PERSON
* (make-person :name "Bobby Beach")

#S(PERSON :NAME "Bobby Beach" :ADDRESS NIL :PHONE-NUMBER 0)
*
```

Tables d'hachage

```
* (setf *x* (make-hash-table))

#<EQL hash table, 0 entries 480AA9B5>
* (gethash 'weight *x*)

NIL
NIL
* (setf (gethash 'weight *x*) 234)

234
* (gethash 'weight *x*)

234
T
*
```

Tables d'hachage

```
* (setf (gethash 11 *x*) nil)

NIL
* (gethash 11 *x*)

NIL
T
*
```

Fonctions de première classe

Un objet est de première classe s'il peut être : la valeur d'une variable, l'argument à un appel de fonction, et renvoyé par une fonction.

```
* #'1+

#<Function 1+ 103744B9>
* (mapcar #'1+ '(4 3 4 8))

(5 4 5 9)
* (reduce #'max '(4 3 5 8))

8
* (funcall (complement #'<) 3 5)

NIL
*
```

Fonctions anonymes

```
* (mapcar (lambda (x) (+ x 2)) '(3 4 5 6))

(5 6 7 8)
* (find-if (lambda (x) (> x 5)) '(5 8 3 9 4 2) :from-end t)

9
* (defparameter *l* (list "how" "hi" "are" "you"))

*L*
* (sort *l* (lambda (x y) (< (length x) (length y))))

("hi" "how" "are" "you")
* *l*

("how" "are" "you")
*
```

Structures de contrôle

- Blocs
- Contextes
- Conditionnels
- Itération
- Valeurs multiples
- Sauts non locaux

Blocs

Trois types de bloc: *progn*, *block*, *tagbody*

progn permet d'évaluer des expression en séquence. La valeur de la dernière expression est la valeur de l'expression *progn*

```
* (progn
    (setf *l* '(5 2 4 3))
    (setf *ll* (sort *l* #'<))
    (car *ll*))

2
*
```

Blocs

block est comme un *progn* avec un nom.

```
* (block found
    (print "hello")
    (when (> 3 2)
      (return-from found 23))
    (print "hi"))

"hello"
23
*
```

tagbody est la construction de base pour les blocs, mais n'est jamais utilisé directement par un utilisateur

Contexte

Un contexte est un bloc plus des variables locales

```
* (let ((len (length *features*))
        (fun (complement #'<)))
    (funcall fun len 10))

T
* (let* ((len (length *features*))
         (min (min len 5)))
    (make-array (list min)))

#(0 0 0 0 0)
*
```

Contexte

la dernière expression est équivalente à :

```
* (let ((len (length *features*)))
    (let ((min (min len 5)))
      (make-array (list min))))

#(0 0 0 0 0)
*
```

Conditionnels

```
(if (f ...)
    (g ...)
    "hello")

(when (f ...)
  (g ...)
  (setf x 23)
  (+ x y))

(unless (f ...)
  (g ...)
  (setf x 23)
  (+ x y))
```