

## Programmation 3 : feuille 10

*Définition et utilisation des macros  
Augmentation et adaptation du Common Lisp de base*

### Exercice 10 .1

Tester quelques macros vues en cours. Comprendre le phénomène de capture des variables.

### Exercice 10 .2

Utiliser `macroexpand-1` et `macroexpand` pour observer l'expansion de diverses macros : `defun`, `defvar`, `defparameter`, `setf`, `cond`, `do`, ...

### Exercice 10 .3

1. Définir une macro `defpower` telle que l'évaluation d'une expression de la forme `(defpower nom n)` définisse la fonction `nom` comme l'élévation à la puissance `n` :

```
* (defpower cube 3)
cube
```

```
* (cube 3)
27
```

```
* (macroexpand-1 '(defpower carre 2))
(DEFUN CARRE (X) (* X X))
T
```

2. Que se passe-t-il si un utilisateur s'amuse à faire évaluer une expression genre `(defpower p5 (+ 2 3))`? Pourquoi?

### Exercice 10 .4

Définir une structure de contrôle similaire au "repeat-until" du langage Pascal.

```
(defun essai (n)
  (repeat-until (<= n 0)
    'ok
    (format T "n=~d%" n)
    (decf n)))
```

```
(essai 6)
;n=6
;n=5
;n=4
;n=3
;n=2
;n=1
;ok
```

## Exercice 10 .5 *Macros avancées*

1. Écrire la fonction `circ` telle que `circ (l)` rend la liste `l` circulaire.
2. Écrire une macro `def-sequencer (nom l)` qui (re)définit la fonction `nom` laquelle énumère ad-infinitem les éléments de la liste `l` rendue circulaire. Par exemple, l'appel  
`(def-sequencer sequencer '(do mi sol))`  
donnera une fonction `sequencer` telle que  
`(list (sequencer)(sequencer)(sequencer)(sequencer)(sequencer))`  
s'évalue en  
`(do mi sol do mi)`.
3. Écrire la macro `with-sequencer (nom l &body body)` en s'inspirant des macros `with-...` déjà rencontrées.

On aura par exemple :

```
CL-USER> (with-sequencer pom-pom '(sol sol sol mi fa fa fa re)
           (loop
             repeat 10
               collect (pom-pom)))
```

```
(SOL SOL SOL MI FA FA FA RE SOL SOL)
```