# Les instructions composées

En programmation il existe plusieurs types d'instructions composées.

## Les instructions conditionnelles

### • COND

L'instruction conditionnelle COND permet de choisir l'exécution d'une action ou une autre en fonction de la valeur d'une expression.
Cette instruction permet de sélectionner une condition parmi plusieurs expressions. Dès que la valeur de l'expression est trouvée, les autres expressions sont sautées.

Sa syntaxe est :

```
(Cond
        (expression 1 à évaluer)
        (expression 2 à évaluer)
        (expression n à évaluer)
) ; fermeture de la condition Cond
```

Exemple :

```
(Defun C:Ex_Cond ()
 (setq test (getint "Entrez un nombre: "))
 (cond
        ((= test 0)
          (alert "Test = 0")
        )
          ; compris entre 1 et 9
        ((and (> test 0) (< test 10))
         (alert  "Test est plus grand que 0 et ✂
          inférieur à 10")
        )
  ; autre valeur que 0, 1 à 9
   (T
        (alert "Test est plus grand que 9 ou ✂
        inférieur à 0")
   )
 )
  (princ)
 ); fin du defun
```

La fonction COND est évaluée séquentiellement à partir de la première expression.

### • IF

L'instruction conditionnelle IF permet de choisir l'exécution d'une action ou d'une autre en fonction d'une condition.

Sa syntaxe est :

```
(IF <Condition>
        action vraie
        [action fausse]
) ; fin du if
```

L'action vraie doit se situer de suite après la condition, sur une ligne.
L'action fausse doit se situer de suite après la condition vraie, sur une ligne.

L'action fausse est facultative. Si elle n'existe pas, seule l'action vraie sera exécutée si elle répond à la condition.

```
(IF <Condition>
action vraie
) ; fin du if
```

Exemple

```
(Defun C:Ex_IF ()
(setq test (getint "\nEntrez un nombre: "))
(if (> test 9)
        ; condition vraie
        (alert "Test est plus grand que 9")
        ; condition fausse
        (alert "Test est plus petit que 9")
)
(princ)
); fin defun
```

Dans le cas où l'expression vraie ou fausse ne peut pas tenir sur une seule ligne, il est possible d'augmenter le nombre de lignes en faisant appel à la fonction PROGN.

Les syntaxes sont :

```
(IF <Condition>
(progn
   action vraie
   action vraie
   action vraie
); fin du progn
   [action fausse]
) ; fin du if
```

```
(IF <Condition>
 action vraie
 (progn
    action fausse
    action fausse
 ) ;fin du progn
 ) ; fin du if
```

```
(IF <Condition>
(progn
 actions vraies...
) ; fin du progn
(progn
 actions fausses...
) ; fin du progn
) ; fin du if
```

## Les instructions répétitives

### • REPEAT

Cette fonction permet d'évaluer un bloc d'instructions répétées jusqu'à une valeur limite.

Sa syntaxe est :

```
(Repeat <nombre de fois>
        expression ...
 ; fin du repeat
```

Exemple :

```
(Defun C:Ex_repeat ()
        (setq test 1)
        ; nombre d'itération
        (setq nbre 10)
                (repeat nbre
                    (alert (itoa test))
                    setq test (1+ test))
                ) ; fin de la boucle repeat
        (princ)
        ); fin du defun
```

Il n'est pas possible de modifier le nombre d'itérations en cours de traitement.

### • WHILE

Dans une instruction WHILE, l'expression est évaluée à chaque début du bloc d'instructions.
Tant que l'expression de condition est vraie, le traitement boucle. Il y a sortie du bloc d'instructions lorsque l'expression de condition est fausse.

La syntaxe est :

```
(WHILE <condition>
        action ...
) ; fin du while
```

Exemple :

```
(Defun C:Ex_WHILE ()
(setq test 0)
(while (< test 10)
        ; test augmente de 1
```

```
(setq test (1+ test))

) ; fin du while
(alert  "A la sortie du while, test = 10))
(princ)
) ; fin du defun
```

Dans le cas où la condition n'est pas vérifiée, l'instruction répétitive WHILE est sautée.

**Attention :**

Avec cette fonction vous risquez de créer une boucle sans fin. En effet, dans le traitement du bloc, il se peut que la condition ne soit plus satisfaite. La seule manière de mettre fin à cette boucle est d'appuyer sur la touche ESC (Echap). Tout le programme est alors arrêté.