

# Les Cahiers d'AutoCAD®

www.dominique-vaquand.com

53

La revue technique pour les utilisateurs d'AutoCAD

12 €

ISSN 1627-0576

## Spécial initiation au langage AutoLisp

C'est dès les premières versions d'AutoCAD que le langage AutoLisp a été intégré au logiciel. L'avantage de ce langage est qu'il peut être directement utilisé sur la ligne de commande, contrairement aux autres langages tels que VBA, DotNet ou ARX.

Pendant longtemps il a été considéré comme un langage de second ordre. Même s'il ne peut être utilisé qu'à l'intérieur d'AutoCAD, il n'en reste pas moins un langage puissant pour le traitement de tous les objets de la base de données d'un dessin DWG.

Avec le temps, on aurait pu penser qu'AutoLisp disparaîtrait, pour laisser place à de nouveaux langages, mais il n'en est rien, même si depuis plusieurs années, il n'a pas vraiment évolué.

Dans ce numéro spécial des cahiers nous allons vous faire découvrir comment utiliser ce langage au sein d'un dessin DWG, comment créer, modifier des objets pour automatiser des tâches répétitives ou créer des commandes utilitaires.

Nous allons, par une approche progressive, vous apprendre à mettre en oeuvre ce langage et à vous donner envie de l'utiliser. N'ayez aucune crainte, même si au début cela peut vous paraître complexe, vous y trouverez vite un intérêt sans pour autant créer une "usine à gaz".

### Le premier pas

Pour ce premier pas nous allons tout simplement créer un cercle de rayon de 10 unités, que nous placerons à une coordonnée de 0,0.

Sur la ligne de commande AutoCAD, tapez ce qui suit, puis validez.

(command "cercle" 0,0 10)

Au sommaire du numéro :

Le premier pas - Des opérations - Définir une nouvelle commande - Mémoriser des valeurs - Poser des questions - Premier exemple pratique - Les variables systèmes - Des commentaires - L'action d'une commande - Deuxième exemple pratique - Charger une commande - Manipuler des objets - Troisième exemple pratique - Quatrième exemple pratique - Cinquième exemple pratique - Sixième exemple pratique - Une commande avec des arguments - Septième exemple pratique - Conclusion - Conseil - Abonnement.

Voilà, si vous n'avez pas fait d'erreur, vous venez de créer votre premier programme AutoLISP.

Comme vous pouvez le constater, une fonction AutoLisp commence par une parenthèse ouverte et se termine par une parenthèse fermée.

Juste derrière la parenthèse ouverte on retrouve le nom de la fonction. Dans notre exemple, la fonction correspond au lancement d'une commande, donc : **(Command**

Ensuite, on indique le nom de la commande à exécuter, nom que l'on place entre des guillemets. Ici on souhaite lancer la commande **"Cercle"**.

Après cela on suit l'ordre des questions posées par la commande lancée. La commande cercle demande en premier le point centre du cercle et ensuite son rayon ou son diamètre.

Le centre est noté par une coordonnée X,Y : **"0,0"** (notez que la coordonnée est écrite entre des guillemets). Le rayon est noté par une valeur numérique : 10 (le rayon étant une valeur numérique, sa valeur n'est pas placée entre des guillemets).

En finalité, pour tracer un cercle de 10 unités de rayon à un point centre de 0,0, la syntaxe AutoLisp se traduit par :

**(command "cercle" 0,0 10)**

Notez que vous pouvez écrire le code en caractères minuscules ou en majuscules, ou panacher si cela vous fait plaisir. Pour une meilleure compréhension la commande est écrite sur une seule ligne, mais elle aurait pu tout aussi bien être écrite sur plusieurs lignes, comme ceci :

```
(command
    "cercle"
    "0,0" 10
)
```

Notez aussi que pour la coordonnée du centre du cercle, nous n'avons pas indiqué la valeur de Z. Cette valeur est prise par défaut si elle n'est pas indiquée, sinon il aurait fallu la signifier comme ceci :

**(command "cercle" 10,5,12 10)**

Ici le cercle de 10 unité de rayon est tracé à la coordonnée "10,5" (10 dans les X et 5 dans les Y) et une valeur Z égale à 12 unités.

## Des opérations

Nul besoin d'aller chercher la calculatrice pour faire quelques petites opérations mathématiques. Avec AutoLISP on peut très bien effectuer une opération dans une commande.

Admettons de devoir décaler une ligne au quart d'une longueur de 129 unités. Un calcul de tête n'est pas évident et puis l'erreur est facile,. Par AutoLISP il suffit simplement de poser l'opération :

Commande: DECALER  
 Paramètres courants: Effacer source=Non Calque=Source OFFSETGAPTYPE=0  
 Spécifiez la distance de décalage ou [Par/Effacer/Calque] <Par>: (/ 129 4.0)

Notez la syntaxe de l'opération; au début on ouvre la parenthèse, ensuite on place la fonction, (ici le symbole de la division), on sépare les arguments par des espaces pour écrire la valeur à diviser et le diviseur, sans oublier de fermer la parenthèse. Enfin on valide l'opération.

On pourrait même s'amuser à écrire des opérations dans des opérations. Par exemple, dans le décalage précédent on pourrait dire que la valeur est égale au tiers du quart de 129, auquel on ajoute PI (3.14159...), et on retranche 12.57, c'est à dire :

$$((129 / 4) / 3) + \text{PI} - 12.57$$

Traduite en AutoLISP, cette opération devient :

$$(- (+ (/ (/ 129 4.0) 3.0) \text{PI}) 12.57)$$

Cela donne comme résultat : 1.32159.

Que pouvons-nous constater de cette syntaxe ?

Que l'opération démarre toujours à partir de l'expression la plus imbriquée de la liste. AutoLISP calcule donc en premier  $(/ 129 4.0)$ , puis ce résultat est divisé par 3 et ainsi de suite.

Notez qu'AutoLISP n'a en mémoire qu'une constante : PI, qui est égal à 3.14159.

Si vous avez un doute, tapez donc : !PI, puis validez.

## Définir une nouvelle commande

Quel est l'intérêt de taper à chaque fois tout le code d'une commande ?

Aucun me direz-vous. L'intérêt majeur de la programmation est de pouvoir mémoriser une ou des actions que l'on pourra utiliser le moment voulu et non pas d'écrire à chaque fois l'action à traiter.

Reprenons notre cercle placé à la coordonnée "0,0" et mémorisons trois commandes qui créeront un cercle de rayon de : 10, 20 et 30 unités.

Ces commandes auront pour nom : C10, C20 et C30

Tapez textuellement ceci :

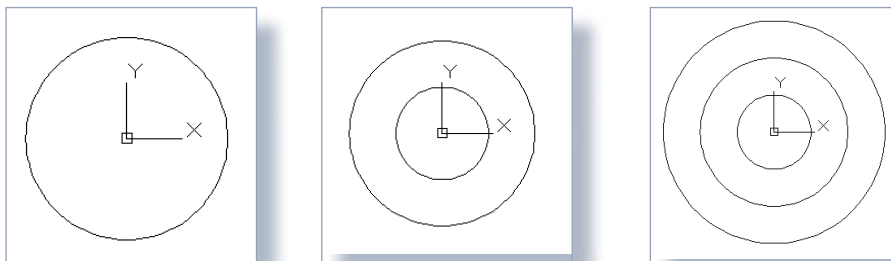
```
(Defun c:C10 () (Command "cercle" "0,0" 10) )
(Defun c:C20 () (Command "cercle" "0,0" 20) )
(Defun c:C30 () (Command "cercle" "0,0" 30) )
```

Vous venez de créer et de mémoriser, temporairement dans votre session de dessin, trois commandes.

Tapez au clavier : C20.

Voyez le cercle de rayon 20 apparaître au centre de la coordonnée : "0,0"

Tapez maintenant : C10, puis C30.



Tant que ces commandes ne seront pas redéfinies, et tant que vous resterez dans ce dessin, ces commandes seront accessibles. Si vous quittez votre dessin, même en le sauvegardant, ces commandes seront perdues. Elles seront perdues car elles n'ont pas été mémorisées en tant que fichier. Ne soyez pas impatient, vous verrez plus loin comment enregistrer votre propre commande dans un fichier.

C'est la fonction **Defun** qui définit une commande AutoLISP. La syntaxe est la suivante :

```
(Defun <type de commande><Nom de la commande> (<Arguments / Noms des variables locales>)
<Action ....>
)
```

*<Type de commande>*

Le caractère : "c:" devant le nom de la commande ne signifie pas l'unité de votre poste mais indique simplement que cette commande est une commande directe sans argument.

Si ce caractère n'existe pas, la commande peut contenir des arguments.

*<Nom de la commande>*

Le nom de la commande ne doit pas contenir des espaces, ni des signes particuliers tels que "/" " ", ...

*<Arguments>*

Si la commande le permet, son appel sera placé dans une liste, par exemple (AfficheInfo), et pourra posséder des arguments, par exemple (AfficheListe Obj 1).

*<Variables locales>*

Si les variables ne sont pas déclarées en local, elles sont alors dites globales, c'est-à-dire que leur valeur pourra être récupérée et traitée par d'autres fonctions en dehors de la commande.

*<Action ...>*

L'action correspond au déroulement du programme à l'intérieur de la commande.

## Mémoriser des valeurs

Pour mémoriser une valeur on utilisera la fonction **SETQ**. Cette fonction initialisera le nom d'une variable qui pourra contenir une chaîne de caractères, une valeur entière ou réelle, une liste et même le contenu d'une autre variable.

Le nom de la variable ne doit pas commencer par un chiffre, ni contenir des espaces ou des caractères spéciaux et autant que possible ne doit pas avoir le même nom qu'une fonction AutoLISP (si, cela est possible, mais le programme risque de ne plus fonctionner par la suite).

Tapez au clavier ceci :

```
(Setq A 10)
(Setq B "Bienvenue")
(Setq C (list 0 10))
```

Ces trois fonctions mémorisent dans la variable "A" la valeur numérique 10, dans la variable "B" la chaîne de caractère "Bienvenue" et dans la variable "C" la coordonnée d'un point où X vaut 0 et Y vaut 10. Pour retrouver la valeur contenu dans une variable tapez le signe "!" (point d'exclamation) devant le nom de la variable. Par exemple :

```
!a retourne la valeur 10
!b retourne la chaîne "Bienvenue"
!C retourne 0,0
!d retourne NIL. NIL signifie que la variable n'a pas été initialisée et ne contient aucune valeur. Ne pas confondre la valeur NIL par "" qui signifie : vide.
```

## Poser des questions

Il faut, dans un programme, pouvoir à un moment donné entrer des valeurs. Dans ce cas le processus de traitement doit s'arrêter et attendre qu'on ait répondu à la question qui nous est posée, puis continuer la tâche. AutoLISP possède des fonctions qui permettent, d'interrompre le traitement en cours dans l'attente d'une réponse à une question posée. Le nom de ces fonctions commence par "GET", les plus utilisées sont :

**GETPOINT** pour définir la coordonnée (X,Y ou X,Y,Z) d'un point. Cette valeur pourra être entrée au clavier ou provenir d'un point cliqué dans la zone graphique d'AutoCAD.

**GETINT** pour entrer une valeur numérique entière, c'est-à-dire une valeur ne contenant pas de décimale, par exemple : 10, 200 mais pas 10.1 ni 200.23.

**GETREAL** pour entrer une valeur numérique réelle, c'est-à-dire une valeur pouvant posséder des décimales, par exemple : 10.1 ou 200.23 ou même 300 qui sera converti en valeur réelle 300.00.

**GETSTRINT** pour entrer une chaîne de caractères par exemple "Bienvenue".

**GETANGLE** pour entrer la valeur d'un angle. Cette valeur sera exprimée en radian. On verra plus loin comment les valeurs angulaires sont gérées en AutoLISP.

**GETDIST** pour entrer une valeur, entière ou réelle ou la distance entre deux points cliqués dans la zone graphique d'AutoCAD.

Il existe bien d'autres fonctions permettant de saisir une valeur, nous les découvrirons au fur et à mesure de l'évolution de ce numéro.

Si vous tapez au clavier : (**GETINT**), AutoCAD attendra que vous entriez une valeur. Employée toute seule cette fonction n'a aucun sens. Il faut l'associer avec une fonction qui permettra de mémoriser la valeur que l'on aura entrée. Tapez au clavier :

```
(SETQ Longueur (GETINT))
```

C'est mieux, car maintenant la valeur que l'on aura entrée sera mémorisée dans la variable "**Longueur**", mais ce n'est toutefois pas suffisant, car il n'y a aucun message qui nous dit ce que l'on doit faire. Doit-on cliquer un point, entrer une chaîne de caractères, entrer une valeur et puis à quoi sera destinée la valeur que l'on aura tapée ?

Soyons plus explicite et ajoutons un message à cette fonction. Tapez au clavier :

```
(SETQ Longueur (GETINT "\nEntrer la longueur du rectangle : "))
Entrez : 100
```

Bien, maintenant on sait que la valeur que l'on tapera correspondra à la longueur du rectangle.

Après avoir entré la valeur et après avoir validé celle-ci, on pourra vérifier si la variable "Longueur" contient bien une valeur en tapant au clavier :

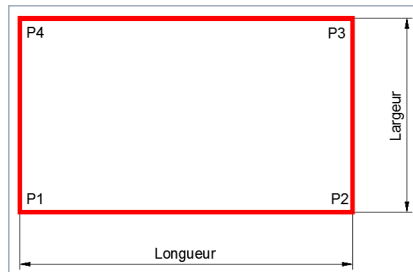
```
!Longueur
100
```

Notez que le message doit être écrit entre des guillemets. Pour obtenir un saut de ligne lors de l'affichage du message on fait précéder la chaîne par : "\n".

Maintenant passons à la pratique.

## Premier exemple pratique

Soit à créer une forme rectangulaire dont on définit la longueur et la largeur. Le point de départ est placé au point bas gauche du rectangle.  
Cette nouvelle commande portera le nom de R1.



```

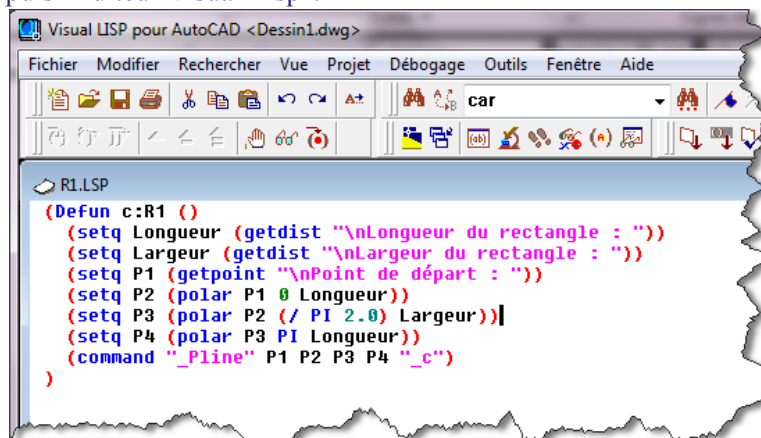
1 (Defun c:R1 ()
2   (setq Longueur (getdist "\nLongueur du rectangle : "))
3   (setq Largeur (getdist "\nLargeur du rectangle : "))
4   (setq P1 (getpoint "\nPoint de départ : "))
5   (setq P2 (polar P1 0 Longueur))
6   (setq P3 (polar P2 (/ PI 2.0) Largeur))
7   (setq P4 (polar P3 PI Longueur))
8   (command "_Pline" P1 P2 P3 P4 "_c")
9 )

```

Pour écrire ce code, nous allons nous servir de l'éditeur Visual Lisp. Pour accéder à cet éditeur vous pouvez soit taper la commande VLIDE, soit depuis le menu "Outils", sélectionner



"AutoLisp" puis "Editeur Visual Lisp".



Une fois que vous avez tapé le code, sauvegardez le fichier sous le nom de : R1.LSP.

Pour le charger dans AutoCAD, cliquez sur le bouton "Charger le programme" dans l'éditeur Visual Lisp. Pour revenir sur AutoCAD, cliquez sur le bouton "Revenir à AutoCAD".



Sur la ligne de commande AutoCAD, tapez : R1.

Le programme est lancé et il vous demande de définir la longueur puis la largeur et enfin le point de départ du rectangle. Si vous n'avez pas fait de faute dans l'écriture du code, un rectangle doit normalement apparaître.

```

Commande: R1
Longueur du rectangle : 100
Largeur du rectangle : 50
Point de départ : nil
Commande:
    
```

Décomposons le contenu de ce programme.

*Ligne 1 : Début du programme avec R1 comme nom.*

*Le "c:" ne définit pas l'unité du disque mais indique que cette commande est une commande directe sans paramètre.*

```
(Defun c:R1 ())
```

*Ligne 2 : Cette ligne demande à l'utilisateur une valeur qui sera mémorisée dans la variable globale "Longueur".*

```
(setq Longueur (getdist "\nLongueur du rectangle : "))
```

*Ligne 3 : Cette ligne demande à l'utilisateur une valeur qui sera mémorisée dans la variable globale "Largeur".*

```
(setq Largeur (getdist "\nLargeur du rectangle : "))
```

*Ligne 4 : Cette ligne demande de définir un point, soit en cliquant avec le bouton gauche de la souris, en en tapant au clavier la coordonnée "X,Y", voire "X,Y,Z" si vous travaillez en 3D. Cette coordonnée sera mémorisée dans la variable globale "P1".*

```
(setq P1 (getpoint "\nPoint de départ : "))
```

*Ligne 5 : Cette ligne, très intéressante, permet de calculer grâce à la fonction "POLAR" la coordonnée "P2" en fonction de celle de "P1", de l'angle de direction (exprimé en radian) et de la longueur. Cette coordonnée sera mémorisée dans la variable globale "P2".*

```
(setq P2 (polar P1 0 Longueur))
```

*Ligne 6 : Cette ligne utilise le même principe que pour le calcul de "P2".*

(setq P3 (polar P2 (/ PI 2.0) Largeur))

*Ligne 7 : Cette ligne utilise le même principe que pour le calcul de "P2".*

(setq P4 (polar P3 PI Longueur))

*Ligne 8 : Cette ligne lance la commande "Polyligne", écrite ici en anglais "\_PLINE", en donnant les différents points de construction "P1, P2, P3, ". A la fin de cette commande on lui indique par l'option "\_c" de se clore.*

(command "\_Pline" P1 P2 P3 P4 "\_c")

*Ligne 9 : On n'oublie pas de terminer le programme par la parenthèse fermante ")". Rappelez-vous que le langage LISP utilise le principe de listes qui doivent être ouvertes au départ et fermées à la fin.*

)

Cette commande, telle qu'elle est, fonctionne, mais présente toutefois quelques imperfections. Si vos accrochages aux objets sont activés et que les dimensions du rectangle s'avèrent trop petites, la forme rectangulaire risque d'être d'une autre forme.

Dans ce cas, il convient, avant que la commande de la ligne 8 soit exécutée, de désactiver les accrochages aux objets. Nous allons voir comment faire cela au chapitre suivant.

## Les variables systèmes

Une variable système, comme son nom l'indique, mémorise, dans la dessin ou dans la base de registre de votre poste, des valeurs ou des informations.

Grâce à la fonction "GETVAR", on va pouvoir lire le contenu de la variable. Avec la fonction "SETVAR" on va pouvoir modifier la valeur de la variable. Cela n'est pas totalement vrai, car certaines variables systèmes sont en lecture seules.

Tapez au clavier : (GETVAR "DWGNAME")

AutoCAD vous répond en affichant le nom du dessin courant. Cette variable système est en lecture seule. De ce fait, si vous tapez la commande (SETVAR "DWGNAME") AutoCAD affichera un message d'erreur qui ne sera pas toujours bien clair !

Pour obtenir la liste complète des variables systèmes, tapez la commande **MODIFVAR**.

```

Commande:
MODIFVAR Entrez le nom de la variable ou [?]: ?

Entrez les variables à répertorier <*>:

3DCONVERSIONMODE 1
3DDWFPREC 2
3DSELECTIONMODE 1
ACADLSPASDOC 0
ACADPREFIX "C:\Users\dv\appdata\roaming\autodesk\autocad 2012 -
french\r..." (lecture seule)
ACADVER "18.2s (LMS Tech)" (lecture seule)
ACISOUTVER 70
ACTPATH ""
ACTRECORDERSTATE 0 (lecture seule)
ACTRECPATH "C:\Users\dv\appdata\roaming\autodesk\autocad 2012 -
french\r..."
ACTUI 6
AFLAGS 16
ANGBASE 0
ANGDIR 0
ANNOALLVISIBLE 1
ANNOAUTOSCALE 4
ANNOTEXTVERING 0
    
```



Dans notre cas, la variable système qui gère les accrochages aux objets se nomme "OSMODE". Elle peut prendre différentes valeurs en fonction des accrochages définis par défaut. Si sa valeur bascule à 0 (zéro), les accrochages seront annulés.

```

1 (Defun c:R1 ()
2   (setq Longueur (getdist "\nLongueur du rectangle : "))
3   (setq Largeur (getdist "\nLargeur du rectangle : "))
4   (setq P1 (getpoint "\nPoint de départ : "))
5   (setq P2 (polar P1 0 Longueur))
6   (setq P3 (polar P2 (/ PI 2.0) Largeur))
7   (setq P4 (polar P3 PI Longueur))
8   (setq AccrochagesCourants (getvar "OSMODE"))
9   (setvar "OSMODE" 0)
10  (command "_Pline" P1 P2 P3 P4 "_c")
11  (setvar "OSMODE" AccrochagesCourants)
12 )

```

La ligne 8 mémorise la valeur de la variable système "OSMODE" dans notre variable "AccrochagesCourants".

La ligne 9 bascule la valeur de la variable système "OSMODE" à 0.

La ligne 11 bascule la valeur de la variable système "OSMODE" à sa valeur initiale après le tracé de la forme rectangulaire.

Dans vos différents programmes vous serez amené à utiliser des variables systèmes, soit pour en récupérer leur valeur, soit pour les modifier, soit les deux à la fois si cela est permis.

## Des Commentaires

Que serait un programme s'il ne contenait aucun commentaire ? Il serait difficile à lire et à comprendre. Ces commentaires permettront donc d'expliquer à quoi est destinée telle commande, telle ligne de code, à décrire ce qu'est censée faire telle condition, etc. Il est important, même pour vous, de documenter un programme.

Pour mettre une ligne en commentaire, on la fait précéder d'un caractère ";" (point-virgule), voire de deux ou trois. L'éditeur Visual Lisp en met trois. Il ne faut toutefois pas trop exagérer !

```

; Mémorise la valeur des accrochages courants
(setq AccrochagesCourants (getvar "OSMODE"))
; Bascule les accrochages à zéro
(setvar "OSMODE" 0)
(command "_Pline" P1 P2 P3 P4 "_c")
; Bascule les accrochages à la valeur d'origine
(setvar "OSMODE" AccrochagesCourants)
)

```

Un commentaire peut être placé à la fin d'une fonction. Tout ce qui se trouvera après le ";" sera considéré comme un commentaire. Nous aurions bien pu documenter la ligne 9 comme ceci :

```
(setvar "OSMODE" 0) ; Bascule les accrochages à zéro
```

## L'action d'une commande

Dans la commande "R1", nous nous sommes servis de la commande POLYLIGNE pour tracer la forme rectangulaire. Nous aurions pu tout aussi bien utiliser la commande LIGNE pour un effet identique. Cette forme aurait été alors constituée de 4 objets au lieu d'un seul pour la polyligne.

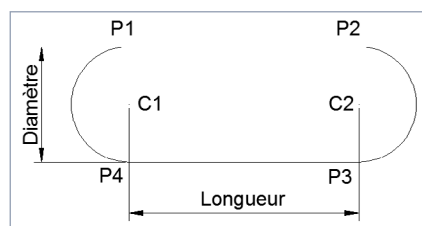
Dans l'utilisation d'une commande AutoCAD il sera nécessaire de connaître toutes ses questions, toutes ses options ainsi que leur ordre.

Par exemple, pour la commande "LIGNE" il faudra définir un point de départ, d'autres points et lui indiquer qu'elle se termine. Pour la commande "INSERER" il faudra définir le nom du bloc, le point d'insertion, le facteur d'échelle dans les X et Y ainsi que l'angle de rotation, sans oublier les attributs si le bloc en contient.

C'est au fur et à mesure de l'action des commandes que vous apprendrez toutes ces options.

## Deuxième exemple pratique

Pour concrétiser les différents points précédents, voyons comment créer un trou oblong dont la longueur correspond à deux fois la valeur du diamètre. Ce trou oblong sera construit par des lignes et des arcs de cercle. Le point de départ sera placé au centre de l'un des arcs. Les points de construction sont donnés par la figure ci-dessous.



```

1 (defun c:TrouOblong ()
2
3   ; Définition des données
4   (setq PointDeDepart (getpoint "\nPoint de départ : "))
5   (setq Diametre (getreal "\nDiamètre du trou oblong : "))
6
7   ; Définition des points
8   (setq C1 PointDeDepart)
9   (setq C2 (polar C1 0 (* diametre 2)))
10  (setq P1 (polar C1 (/ PI 2.0) (/ diametre 2.0)))
11  (setq P2 (polar P1 0 (* diametre 2)))
12  (setq P3 (polar P2 (* 1.5 PI) diametre))
13  (setq p4 (polar P1 (* 1.5 PI) diametre))
14
15  ; Tracé de la forme
16  (command "_Arc" "_c" C1 P1 "_A" 180)
17  (command "_Line" P4 P3 "")
18  (command "_Arc" "_C" C2 P3 "_A" 180)
19  (command "_Line" P2 P1 "")
20
21  (princ)
22 )

```

Détaillons les différentes lignes :

- Ligne 1 : Cette ligne déclare la nouvelle commande qui se nomme ici "TrouOblong".
- Ligne 4 : Cette ligne stoppe le programme pour nous demander le point de départ du trou oblong. Ce point pourra être défini soit par un clic, soit en entrant une coordonnée "X,Y". Dans notre exemple, le point de départ correspondra au point C1. Cette coordonnée sera mémorisée dans la variable "PointDeDepart".
- Ligne 5 : Cette ligne stoppe le programme pour nous demander d'entrer une valeur qui correspondra au diamètre du trou. Cette valeur sera mémorisée dans la variable "Diametre".
- Ligne 8 : On définit la variable "C1" qui contiendra la valeur de la variable "PointDeDepart".
- Ligne 9 : On définit la variable "C2" qui contiendra une coordonnée correspondant à une distance de deux fois la valeur du diamètre selon une direction 0. Cette direction est exprimée non pas en degrés mais en radian (Autolisp calcule les angles en radian).
- Ligne 10 : On définit le premier point de construction de l'arc qui sera mémorisé dans la variable P1. La position de P1 est définie par rapport au point C1, selon une direction de (/ PI 2) radian, soit 90° (PI radian équivaut à un angle de 180°).
- Ligne 11 : On définit le point P2 par rapport au point P1. Ce point P2 correspondra au point final de la ligne entre les points P1 et P2.
- Ligne 12 : On définit le point P3 qui, comme le point P1, correspondra au point de départ de construction de l'arc. La direction (\* 1.5 PI) radian correspond à une direction de 270°.
- Ligne 13 : On définit le point P4 par rapport au point P3. Ce point P4 correspondra au point final de la ligne entre les points P3 et P4.
- Ligne 16 : On lance la commande ARC avec l'option "Centre" où l'on définit le point centre C1, le point de départ P1 et son angle d'ouverture 180. Remarquez que l'angle est exprimé ici en degré. Ce type correspond à l'unité d'angle définie par la commande UNITS.
- Ligne 17 : On lance la commande LIGNE qui partira du point P1 pour aboutir au point P2. On n'oublie pas de terminer la commande LIGNE par un double guillemet, ou double quote, qui correspond à l'action de la touche de validation (ENTREE).
- Ligne 18 : Même principe que la ligne 10 pour la construction de l'autre arc.
- Ligne 19 : Même principe que la ligne 11 pour la construction de la ligne P3 à P4.
- Ligne 21 : Avant de terminer la programme, on annule l'affichage de la valeur de la dernière ligne de code (ici la ligne 13) renvoyée par le programme. Cette ligne n'est pas obligatoire mais elle fait plus pro !
- Ligne 22 : On pense à fermer la parenthèse correspondant à la fin du programme.

Sauvegardez ce programme sous le nom de fichier "TrouOblong.lsp".

Pour charger le programme, tapez sur la ligne de commande AutoCAD la syntaxe suivante : (LOAD "TrouOblong").

Pour lancer le programme tapez : TrouOblong.

## Charger une commande dans AutoCAD

Il existe plusieurs méthodes pour charger une commande AutoLISP dans AutoCAD.

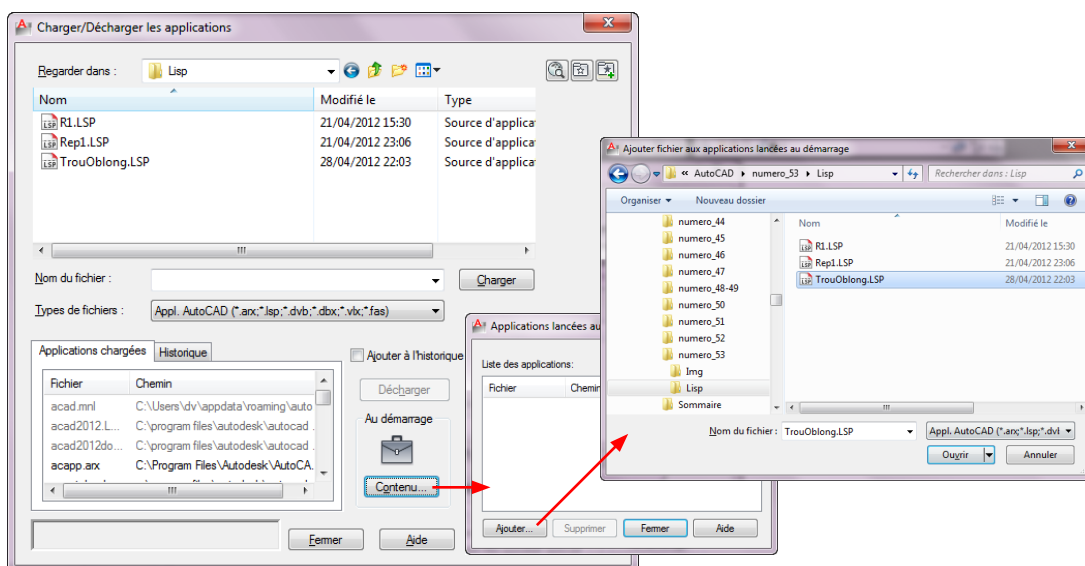
- La première et la plus simple est d'utiliser le Glisser/Déposer. Pour cela il suffit de sélectionner le fichier LSP depuis l'explorateur de fichiers Windows et tout en le faisant glisser, le déposer dans la partie graphique d'AutoCAD.
- La deuxième méthode consiste à taper sur la ligne de commande AutoCAD la ligne suivant :  
(LOAD "Nom du Programme AutoLISP")

Si le fichier LSP n'est pas dans un dossier reconnu par l'environnement d'AutoCAD (Fenêtre des OPTIONS), il faudra faire précéder son chemin du nom du fichier :

(LOAD "C:/Mes Progs/Nom du Programme AutoLISP")

Ces deux premières méthodes doivent être exécutées à chaque ouverture d'un dessin.

- La troisième méthode consiste à utiliser le gestionnaire des applications pour que le programme soit chargé automatiquement au chargement d'AutoCAD.



- La quatrième méthode consiste à mémoriser la syntaxe de la deuxième méthode (LOAD "Nom du Programme AutoLISP") dans l'un des deux fichiers ACAD.LSP ou ACADDOC.LSP qui sera chargé automatiquement au chargement d'AutoCAD.
- La cinquième méthode consiste à écrire dans un fichier MNL la syntaxe de la deuxième méthode (LOAD "Nom du Programme AutoLISP"). Le nom du fichier MNL devra correspondre au nom d'un menu chargé.

Si le programme doit être utilisé occasionnellement, on peut tout aussi bien le charger par l'intermédiaire d'un bouton personnalisé d'une barre d'outils, ou d'une icône depuis la fenêtre de la palette d'outils.

## Manipuler des objets

La manipulation des objets représente une partie très intéressante dans la programmation AutoLISP.

L'accès aux informations d'un objet peut être obtenu de deux façons différentes :

1. Par un accès à ses codes DXF
2. Par un accès aux propriétés VLA qui s'identifient un peu comme du VBA.

Etudions ces deux accès.

### Cas 1 : Accès aux codes DXF

1. Dans votre dessin créez un nouveau calque que vous nommerez "Objet".
2. Tracez sur ce calque un cercle d'un rayon de 10 unités.
3. Tapez la commande : (ENTGET (CAR (ENTSEL)))
4. Sélectionnez le cercle.

Commande: (entget (car (entsel)))

Choix de l'objet: ((-1 . <Nom d'entité: 7ffff605a90>) (0 . "CIRCLE") (330 . <Nom d'entité: 7ffff6039f0>) (5 . "221") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "Objet") (100 . "AcDbCircle") (10 2552.59 1052.68 0.0) (40 . 10.0) (210 0.0 0.0 1.0))

Observez comment sont mémorisées les informations du CERCLE. Elles sont inscrites dans une liste qui contient elle même d'autres listes. Chaque liste interne commence par un code, par exemple (8 . "Objet"). Ces codes correspondent aux codes DXF d'AutoCAD.

Vous pouvez consulter la liste complète de tous ces codes dans l'aide en ligne d'AutoCAD.

Rassurez-vous il n'est nul besoin de connaître tous ces codes par coeur car en règle générale on en traite moins d'une vingtaine.

Chaque code possède une signification

- Le code -1 correspond au code "Entity Name" de l'objet. Ce code est créé automatiquement par AutoCAD mais n'est pas fixe. Il peut changer à la prochaine ouverture du même dessin.
- Le code 0 correspond au type d'objet. Ici nous voyons que l'objet sélectionné est un cercle (0 . "CIRCLE"). Le nom de l'objet est donné en anglais.
- Le code 5 correspond au code de maintien de l'objet. Ce code est exprimé en hexadécimal et est créé automatiquement par AutoCAD. Contrairement au code -1, le code 5 est un code fixe qui ne changera jamais. C'est ce code qui pourra servir à faire des liaisons d'objet avec des programmes externes, entre autres.
- Le code 67 correspond à l'espace sur lequel est créé l'objet. La valeur 0 indique ici que l'objet cercle est créé sur l'espace objet. Une valeur 1 aurait indiqué que l'objet est placé sur l'espace papier.
- Le code 410 indique le nom de l'espace dans lequel l'objet est créé.
- Le code 8 correspond au nom du calque dans lequel l'objet est placé.
- Le code 40 indique le rayon du cercle.
- Le code 10 fournit les coordonnées du centre du cercle.
- Le code 210 correspond au point de vue de l'objet.

## Cas 2 : Accès aux données Visual Lisp

En se référant sur ce même cercle, tapez les commandes suivantes :

1. (vl-load-com)
2. (Setq Obj (car (entsel)))
3. Sélectionnez l'objet cercle
4. (Setq VbaObj (vlax-ename->Vla-Object Obj))
5. (vlax-dump-object VbaObj)

```

Commande: (vl-load-com)
Commande: (setq Obj (car (entsel)))
Choix de l'objet: <Nom d'entité: 7fff605a90>
Commande: (setq VbaObj (vlax-ename->vla-object Obj))
#<VLA-OBJECT IAcadCircle2 000000002bd17f78>
Commande: (vlax-dump-object VbaObj)

; IAcadCircle2: Interface AutoCAD Circle
; Valeurs de propriétés:
; Application (RO) = #<VLA-OBJECT IAcadApplication 00000001400491d8>
; Area = 314.159
; Center = (2552.59 1052.68 0.0)
; Circumference = 62.8319
; Diameter = 20.0
; Document (RO) = #<VLA-OBJECT IAcadDocument 0000000027846f60>
; EntityTransparency = "DuCalque"
; Handle (RO) = "221"
; HasExtensionDictionary (RO) = 0
; Hyperlinks (RO) = #<VLA-OBJECT IAcadHyperlinks 0000000027be2318>
; Layer = "Objet"
; Linetype = "ByLayer"
; LinetypeScale = 1.0
; Lineweight = -1
; Material = "ByLayer"
; Normal = (0.0 0.0 1.0)
; ObjectID (RO) = 42
; ObjectID32 (RO) = 42
; ObjectName (RO) = "AcDbCircle"
; OwnerID (RO) = 43
; OwnerID32 (RO) = 43
; PlotStyleName = "ByLayer"
; Radius = 10.0
; Thickness = 0.0
; TrueColor = #<VLA-OBJECT IAcadAcCmColor 0000000027be2430>
; Visible = -1
T

```

En comparant ces deux cas, vous remarquerez que certaines informations sont communes tout en étant présentées différemment. Par exemple, dans le cas d'un accès aux codes DXF, le nom du calque est indiqué sous la forme (8 . "Objet") tandis que dans le cas d'un accès VLA le nom du calque est indiqué sous la propriété Layer = "Objet". Vous remarquerez aussi que les informations provenant d'un accès VLA sont plus nombreuses. Par exemple pour un objet "Cercle", la propriété de la circonférence est affichée, ce qui évitera de la calculer dans le cas où il serait nécessaire de réaliser des traitements avec.

Selon les opérations que l'on aura à effectuer sur un objet, par exemple le changement de calque, il sera peut-être plus facile de traiter l'objet depuis ses propriétés VLA que depuis ses codes DXF et inversement.

Au regard de ces lignes de code ne pensez pas que la modification des propriétés d'un objet soit si compliquée que cela !

## Troisième exemple pratique

Sélectionner un cercle existant et créer un nouveau cercle de diamètre identique. Le nouveau cercle sera placé sur le même calque que celui du cercle sélectionné.

```

1 ; Début du programme
2 (Defun c:Cercleldem ()
3 ; Sélection d'un objet
4 (setq ObjCercle (entget (car (entsel "\nSélectionner un cercle : "))))
5 ; Récupération de la valeur du cercle
6 (setq RayonDuCercle (cdr (assoc 40 ObjCercle)))
7 ; Récupération du calque du cercle
8 (setq CalqueDuCercle (cdr (assoc 8 ObjCercle)))
9 ; Récupération du calque courant
10 (setq CalqueCourant (getvar "clayer"))
11 ; On se met sur le calque du cercle
12 (command "_layer" "_m" CalqueDuCercle "")
13 ; On trace le cercle
14 (command "_Circle" (getpoint "\nCentre du cercle : ") RayonDuCercle)
15 ; on se remet sur le calque courant
16 (setvar "clayer" CalqueCourant)
17 ; on annule l'affichage retour de la dernière commande
18 (princ)
19 ; Fin du programme
20 )

```

Détaillons ce programme :

La fonction ENTSEL permet de récupérer sous la forme de liste le code "Entity Name" ainsi que la coordonnée du point de sélection de l'objet sélectionné.

(<Nom d'entité: 7ffff605ba0> (2142.35 997.396 0.0))

Pour pouvoir retrouver les propriétés de l'objet il faut isoler la valeur du code "Entity Name". C'est la fonction CAR qui s'en charge.

(car (entsel)) retourne <Nom d'entité: 7ffff605ba0>

La fonction ENTGET va nous permettre de récupérer les différentes valeurs DXF de l'objet.

(entget (car (entsel)))  
 ((-1 . <Nom d'entité: 7ffff605ba0>) (0 . "CIRCLE") (330 . <Nom d'entité: 7ffff6039f0>)  
 (5 . "232") (100 . "AcDbEntity") (67 . 0) (410 . "Model") (8 . "0") (100 . "AcDbCircle")  
 (10 2246.53 724.241 0.0) (40 . 293.33) (210 0.0 0.0 1.0))

La variable "ObjCercle" mémorise la liste de tous les codes DXF de l'objet

La fonction ASSOC associée au numéro de la clé DXF retourne, sous la forme d'une liste "pointée", la valeur de la clé.

(assoc 40 objcercle)  
 (40 . 148.421)

La fonction CDR permet de récupérer le dernier élément de la liste, c'est-à-dire la valeur du rayon.

(cdr (assoc 40 objcercle))  
 148.421

Le code clé 40 correspond au rayon du cercle dans le code DXF d'un objet Cercle.

Par le même principe on récupère le nom du calque sur lequel le CERCLE a été créé.

La fonction GETVAR permet de récupérer la valeur d'une variable système. Ici on récupère la valeur du calque courant mémorisée dans la variable système "CLAYER".

La fonction COMMAND exécute une commande AutoCAD.

La fonction SETVAR permet d'écrire dans la variable système.

Ce programme présente toutefois un défaut. Il n'est pas interdit à l'utilisateur de sélectionner un tout autre objet qu'un cercle. Le code clé 40 qui nous renseigne sur le rayon du cercle n'existe pas sur un objet LIGNE ou un objet TEXTE. Que se passera-t-il si l'utilisateur sélectionne un autre objet qu'un CERCLE ?

Nous vous laissons le soin de ce qui se passera en sélectionnant un autre type d'objet.

Pour éviter cette erreur de sélection, il existe plusieurs méthodes. Pour notre exemple nous allons simplement créer une condition sur l'objet sélectionné en utilisant la fonction IF.

Cette fonction fonctionne selon le principe suivant :

```
(IF condition
  (traitement de la condition vrai)
  (traitement de la condition fausse)
) ; fin de la condition
```

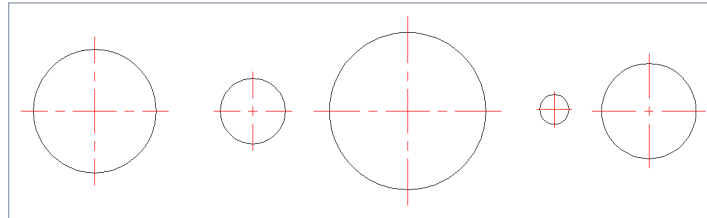
Donc si on modifie quelque peu ce programme, il sera opérationnel quel que soit l'objet sélectionné.

```
1 ; Début du programme
2 (Defun c:CercleIdem1 ()
3 ; Sélection d'un objet
4 (setq ObjCercle (entget (car (entsel "\nSélectionner un cercle : "))))
5 (if (= (cdr (assoc 0 ObjCercle)) "CIRCLE")
6   (progn
7 ; Récupération de la valeur du cercle
8 (setq RayonDuCercle (cdr (assoc 40 ObjCercle)))
9 ; Récupération du calque du cercle
10 (setq CalqueDuCercle (cdr (assoc 8 ObjCercle)))
11 ; Récupération du calque courant
12 (setq CalqueCourant (getvar "clayer"))
13 ; On se met sur le calque du cercle
14 (command "_layer" "_m" CalqueDuCercle "")
15 ; On trace le cercle
16 (command "_Circle" (getpoint "\nCentre du cercle : ") RayonDuCercle)
17 ; on se remet sur le calque courant
18 (setvar "clayer" CalqueCourant)
19 ) ; fin du progn
20 (alert "Vous n'avez pas sélectionné un cercle.")
21 ) ; fin du if
22 ; on annule l'affichage retour de la dernière commande
23 (princ)
24 ; Fin du programme
25 )
26
```



## Quatrième exemple pratique

Créer les deux traits d'axe du cercle sélectionné. Les traits d'axe dépasseront du cercle d'une valeur fixe ou d'un pourcentage de la valeur du rayon et seront placés sur un calque bien précis.



Dans cet exemple, vous découvrirez comment se crée un jeu de sélection d'objets, comment on accède aux objets contenus dans un jeu de sélection, comment on définit des options lors d'une saisie.

```

1 ; Axes sur cercles
2
3 (defun c:AxesCercles ( / OsmodeCourant CalqueCourant ChoixDeLaValeur Valeur ObjSelection
4   NbreObjSelection index CentreDuCercle RayonDuCercle ValeurDeDecalage
5   PT1 PT2 PT3 PT4 )
6
7 ; Mémoire le mode d'accrochage courant
8 (setq OsmodeCourant (getvar "Osmode"))
9 (setvar "osmode" 0) ; Désactive le mode d'accrochage; Mémoire le nom du calque courant
10 (setq CalqueCourant (getvar "Clayer"))
11 ; Crée le calque sur lequel les axes seront placés
12 (command "_Layer" "_M" "AxesCercles" "_LT" "Axes" "" "")
13 ; Définit une option de saisie lors du choix de la valeur
14 (initget 7 "V P Valeur Pourcentage")
15 ; Définit le choix de la valeur
16 (setq ChoixDeLaValeur (getkword "\nValeur ou Pourcentage: "))
17 ; Crée une condition selon le choix de la valeur
18 (cond
19   ; Condition si le choix de la valeur est égale à "V"
20   ((= ChoixDeLaValeur "V")
21     ; controle si la valeur saisie n'est pas nulle, ni négative
22     (initget 7)
23     ; Saisie de la valeur de décalage. La valeur sera de type réelle et pourra être définie par 2 points
24     (setq Valeur (getdist "\nValeur de décalage: "))
25   )
26   ; Condition si le choix de la valeur est égale à "P"
27   ((= ChoixDeLaValeur "P")
28     ; controle si la valeur saisie n'est pas nulle, ni négative
29     (initget 7)
30     ; Saisie de la valeur de décalage. La valeur sera de type réelle.
31     (setq Valeur (getreal "\nPourcentage de décalage: "))
32   )
33 )
34 ; Crée un jeu de sélection qui ne contiendra que des cercles
35 (setq ObjSelection (ssget (list (cons 0 "Circle"))))
36 ; Vérifie si la sélection contient des objets
37 (if (/= ObjSelection nil)

```

```

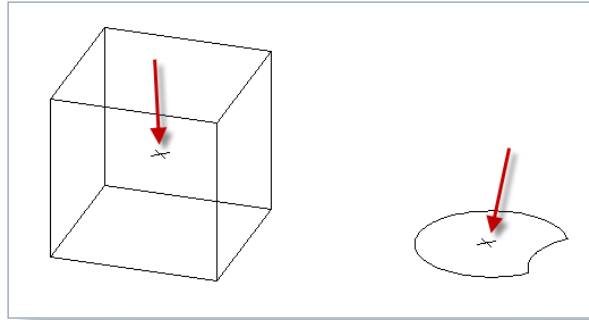
38      (progn ; Agrandi la zone de condition Vrai
39      ; Mémoire le nombre d'objet dans la sélection
40      (setq NbreObjSelection (sslength ObjSelection))
41      ; Définit une variable qui servira d'index pour le jeu de sélection
42      ; dans le jeu de sélection le premier objet est à l'indice 0
43      (setq Index 0)
44      ; crée une boucle sur le nombre d'objet sélectionné
45      (repeat NbreObjSelection
46      ; Mémoire le centre du cercle de l'objet à l'indice de la valeur de l'index
47      (setq CentreDuCercle
48      (cdr (assoc 10
49      (entget (ssname ObjSelection index))
50      )
51      )
52      )
53      ; Mémoire le rayon du cercle de l'objet à l'indice de la valeur de l'index
54      (setq RayonDuCercle
55      (cdr (assoc 40
56      (entget (ssname ObjSelection index))
57      )
58      )
59      )
60      ; Calcule la valeur de décalage selon le choix de la valeur
61      (if (= ChoixDeLaValeur "V")
62      (setq ValeurDeDecalage (+ RayonDuCercle Valeur))
63      (setq ValeurDeDecalage
64      (+ RayonDuCercle
65      (/ (* RayonDuCercle Valeur) 100.0)
66      )
67      )
68      )
69      ; Mémoire les points de création des lignes
70      (setq PT1 (polar CentreDuCercle 0 ValeurDeDecalage))
71      (setq PT2 (polar CentreDuCercle PI ValeurDeDecalage))
72      (setq PT3 (polar CentreDuCercle (/ pi 2.0) ValeurDeDecalage))
73      (setq PT4 (polar CentreDuCercle (* 1.5 pi) ValeurDeDecalage))
74      ; crée les lignes d'axes
75      (command "_Line" PT1 PT2 "")
76      (command "_line" PT3 PT4 "")
77      ; Incrémente la valeur de l'index
78      (setq index (1+ index))
79      ) ; fin de la boucle repeat
80      ) ; Fin du progn de la condition vrai
81      ; Affiche une alerte si aucun cercle n'est sélectionné
82      (alert "Vous n'avez pas sélectionné de Cercles.")
83      ) ; fin de la condition If
84      ; réactive le mode d'accrochage précédent
85      (setvar "osmode" OsmodeCourant)
86      ; réactive le calque précédent
87      (setvar "Clayer" CalqueCourant)
88      (princ)
89      ; fin du programme
90      )

```

## Cinquième exemple pratique

S'accrocher sur le centre de gravité d'un objet 3D ou sur le barycentre d'une surface 2D.

La commande de propriétés mécaniques permet d'obtenir la coordonnée du centre de gravité d'un objet volumique 3D ou d'une région 2D. Cette coordonnée n'étant pas matérialisée par un objet, aussi pour pouvoir localiser ce point ou démarrer un objet dessus, il sera nécessaire de taper manuellement la coordonnées. Ce cinquième exemple pratique localise le centre de gravité en affichant sa coordonnée.



```

1 ; centre de gravité d'un solide
2 ; barycentre d'une région
3
4 (defun c:CGS ()
5
6 ; charge la connexion VBX
7 (vl-load-com)
8
9 ; Sélection d'un objet
10 (setq ObjSelection (car (entsel "\nSélectionner un solide ou une région : ")))
11
12 ; Mémoire le nom de l'objet
13 (setq TypeObjet (cdr (assoc 0 (entget ObjSelection))))
14
15 ; Vérifie que l'objet sélectionné soit binaire un 3DSOLID ou une REGION
16 (if (or (= TypeObjet "3DSOLID") (= TypeObjet "REGION"))
17     (progn
18
19 ; Mémoire la valeur de l'accrochage aux objets
20 (setq Osmode (getvar "Osmode"))
21 ; Désactive les accrochages aux objets
22 (setvar "osmode" 0)
23 ; Mémoire le nom du calque courant
24 (setq CalqueCourant (getvar "clayer"))
25 ; Récupère le nom du calque de l'objet sélectionné et active le calque
26 ;(command "_layer" "_m" (cdr (assoc 8 (entget ObjSelection))) "")
27
28 ; crée un calque nommé "Centre de gravité" et le met courant
29 (command "_layer" "_m" "Centre de gravité" "")
30
31 ; Active le SCU en mode général
32 (command "_ucs" "")

```

```

33
34 ; convertis l'objet en module VBX
35 (setq ObjVBA (vlax-ename->Vla-Object ObjSelection))
36
37 ; Mémorise la coordonnée du centre de gravité au format VBX
38 (setq CentreDeGravite (vla-get-centroid ObjVBA))
39 ; Convertis la coordonnée VBX en coordonnée LISP
40 (setq CentreDeGravite (vlax-safearray->list (vlax-variant-value CentreDeGravite)) )
41
42 ; Crée un objet point que l'on place à la coordonnée du centre de gravité
43 (command "_Point" CentreDeGravite)
44 ; On se replace sur le calque précédent
45 (command "_layer" "_m" CalqueCourant "")
46 ; On réactive le mode d'accrochage aux objets
47 (setvar "osmode" Osmode)
48 ; On revient sur le SCU précédent
49 (command "_ucs" "_p")
50
51 )
52 ; si l'objet sélectionné n'est ni un 3DSOLID, ni une REGION on affiche une alerte
53 (alert "Vous n'avez pas sélectionné un Solide ou une Region.")
54
55 )
56 ; on évite le retour de la dernière valeur
57 (princ)
58 ; fin du programme
59 )

```

Note : Le point n'est pas solidaire du solide ou de la région. Si vous déplacez ou si vous supprimez l'objet, n'oubliez pas d'inclure aussi le point.

---

## Les Cahiers d'AutoCAD

*La revue technique sur AutoCAD* ISSN 1627-0576

Adresse : **Dominique VAQUAND Informatique** - 24, Rue des Icards - BP 33 - 13430 EYGUIERES - France  
Tél : 04. 90.57.96.70 / Fax : 04.90.57.96.23  
Courriel : [contact@dominique-vaquand.com](mailto:contact@dominique-vaquand.com)  
Sites WEB : [www.dominique-vaquand.com](http://www.dominique-vaquand.com)

Directeur de la publication : Dominique VAQUAND  
Correction : Michel P.  
Diffusion : Dominique VAQUAND Informatique

Abonnement : 4 numéros : 40 € TTC  
Au numéro : 12 € TTC

**Les Cahiers d'AutoCAD** est une marque déposée par Dominique VAQUAND Informatique.  
Tous les produits cités dans cette revue peuvent être des marques déposées par leurs propriétaires respectifs.  
Les articles, programmes et fichiers présents avec ce numéro sont livrés en l'état, sans garantie d'aucune sorte.  
Tous droits de reproduction réservés pour tous pays. © Dominique VAQUAND Informatique.  
Les bases d'informations proviennent de recherches sur Internet, du support Autodesk, de l'aide en ligne, de particuliers, de nos connaissances et de nos expériences professionnelles.

## Sixième exemple pratique

Un jeu de sélection permet de mémoriser dans une variable une sélection d'objet. Cette sélection peut être manuelle ou automatique selon le filtre qui lui aura été appliqué. Avec ce jeu, il sera possible d'intervenir sur tous ses objets pour les modifier, par exemple pour les copier, pour modifier certaines propriétés, telles que le calque par exemple, pour récupérer des informations qui permettront par exemple de cumuler toutes les longueurs des lignes ou pour quantifier le nombre de blocs insérés.

Soit à cumuler toutes les longueurs des lignes incluses dans un jeu de sélection.

```

1 ; Total longueur des lignes sélectionnées
2
3 (defun c:TotalLgLignes (/ ObjSelection
4   NombreObjetDansLaSelection
5   Index
6   ObjetDansLaSelection
7   TotalLongueur
8   )
9
10 ; Question
11 (princ "\nSélection des objets : ")
12 ; Sélection des objets
13 (setq ObjSelection (ssget))
14 ; Si la sélection contient des objet alors on continue
15 (if ObjSelection
16   (progn
17     ; Mémoire le nombre d'objet dans la sélection
18     (setq NombreObjetDansLaSelection (sslength ObjSelection))
19     ; Définit une variable index pour récupérer l'objet dans le jeu de sélection
20     ; le premier objet dans le jeu de sélection est positionné à l'index 0
21     (setq Index 0)
22     ; Initialise une variable qui mémorisera et cumulera la distance de chaque ligne
23     (setq TotalLongueur 0)
24
25     ; crée une boucle de répétition sur le nombre d'objet contenu dans le jeu de sélection
26     (repeat NombreObjetDansLaSelection
27       ; Récupère le nom de l'objet positionné sur l'index
28       (setq ObjetDansLaSelection (entget (ssname ObjSelection Index)))
29       ; Si le nom de l'objet est égale à "LINE" alors on récupère la longueur de la ligne
30       (if (= (cdr (assoc 0 ObjetDansLaSelection)) "LINE")
31         ; la longueur de la ligne est égale à la distance entre son point de départ et son point de fin
32         (setq TotalLongueur
33           (+ TotalLongueur
34             (distance
35               ; Point de départ de la ligne
36               (cdr (assoc 10 ObjetDansLaSelection))

```

```

37      ; point de fin de la ligne
38      (cdr (assoc 11 ObjetDansLaSelection))
39      )
40      )
41      )
42      )
43      ; Incrémente la valeur de l'index à +1 pour passer à l'objet suivant du jeu de sélection
44      (setq Index (1+ Index))
45      ; fin du Repeat
46      )
47      ; Affiche, dans la zone de la ligne de commande, la longueur totale des lignes sélectionnées
48      (princ (strcat "\nTotal longueur des "
49      (itoa index)
50      " lignes = "
51      (rtos TotalLongueur 2 (getvar "luprec"))
52      )
53      )
54      ; fin du Progn
55      )
56      ; affiche une boîte d'alerte si aucune sélection n'a été sélectionné
57      (alert "\nVous n'avez sélectionné aucun objet.")
58
59      ; fin du IF
60      )
61      (princ)
62      ; fin du programme
63      )

```

## Une commande avec des arguments

Les commandes telles que nous les avons notées jusqu'à maintenant sont des commandes directes, c'est-à-dire qu'elles peuvent être lancées en tapant simplement leur nom. Une commande peut toutefois posséder des arguments qui seront traités dans le programme.

Voici deux exemples :

Soit à calculer le résultat de la multiplication entre deux valeurs "A" et "B".

```

1  (Defun c:MAB1 ()
2
3  (setq A (getint "\nValeur de A : "))
4  (setq B (getint "\nValeur de B : "))
5
6  (setq Resultat (* A B))
7
8  (Alert (itoa Resultat))
9
10 (princ)
11
12 )

```

Après avoir chargé le programme MAB1, tapez au clavier MAB1.  
 Entrez la valeur 10 pour A et la valeur 2 pour B  
 Le résultat doit retourner : 20.

```

1 (Defun MAB2 (A B)
2
3 (setq Resultat (* A B))
4
5 (Alert (itoa Resultat))
6
7 (princ)
8
9 )
    
```

Après avoir chargé le programme MAB2, tapez au clavier (MAB 10 2).  
 Le résultat retourne 20.

Dans ce cas vous avez dû lancer la commande en plaçant une parenthèse ouverte "(" au début de son nom, en séparant les arguments par des espaces, sans oublier d'ajouter une parenthèse fermante ")" à la fin de la commande .

Ce type de commande est souvent utilisé comme sous-programme ou procédure. Imaginez que depuis des commandes directes vous deviez faire plusieurs fois ce type d'opération. Au lieu d'écrire plusieurs fois les mêmes lignes de code, un sous-programme pourrait être utilisé comme ceci :

```

1 (Defun MonProgrammePrincipal ()
2 ...
3 ; appel du sous programme
4 (MAB2 3 2) ; Exécute le sous-programme MAB2 avec les arguments 3 et 2
5 (MAB2 4 3) ; Exécute le sous-programme MAB2 avec les arguments 4 et 3
6 (MAB2 5 4) ; Exécute le sous-programme MAB2 avec les arguments 5 et 4
7 ...
8 )
    
```

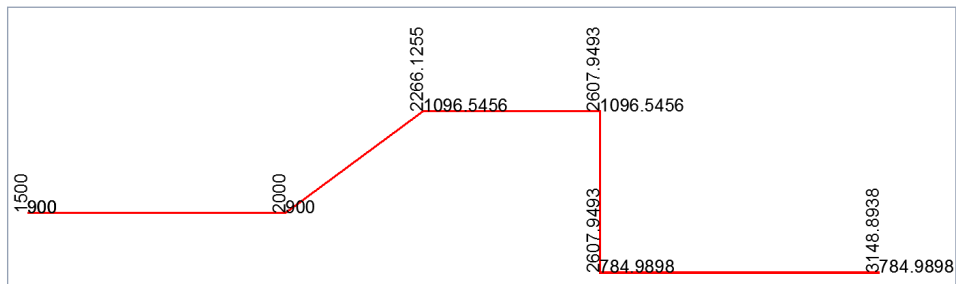
Remarque : AutoLISP ne fait aucun contrôle sur le type des arguments entrés, par contre vous devrez entrer exactement le même nombre d'arguments tel qu'il a été défini dans la commande. Si l'argument est d'un type différent que celui qui doit être traité, par exemple si vous entrez (MAB2 "10" 2) au lieu de (MAB2 10 2), une erreur sera déclarée dès que cet argument sera traité car AutoLISP ne saura pas transformer la chaîne de caractère "10" en une valeur numérique.

**Vous avez besoin d'optimiser au mieux AutoCAD ?  
 Vous souhaitez gagner du temps sur certaines opérations ?  
 Contactez nous pour un devis gratuit.**

## Septième exemple pratique

Soit à placer sur chaque sommet d'une polyligne ses coordonnées X et Y. La polyligne sélectionnée pourra être de type optimisé ou non.

Les coordonnées seront placées dans le même calque que celui de la polyligne.



```

1 ; Coordonnées des sommets d'une polyligne
2
3 (defun c:CSP ( / ObjSelection Index index1 VLAObj NomObj PTDepart ValCoord)
4
5 (vl-load-com)
6
7 (setq Osmode (getvar "osmode"))
8
9 (setvar "osmode" 0)
10
11 (setq CalqueCourant (getvar "Clayer"))
12
13 ; sous-programme pour définir la hauteur du texte si elle n'est pas définie
14 (HauteurTexte)
15 ; Sélection des objets Polygones
16 (setq ObjSelection
17 (ssget
18 (list
19 (cons
20 0
21 "LWPOLYLINE,POLYLINE" )))
22 )
23 ; teste si des objets polygones ont été sélectionnés
24 (if (/= ObjSelection nil)
25 (progn
26 ; incrémente l'index des objets dans la sélection
27 (setq Index 0)
28 ; boucle sur le nombre d'objet trouvé
29 (repeat (sslength ObjSelection)
30 ; récupère les propriétés de l'objet à l'indice ...
31 (setq
32 VLAObj (vlax-ename->vla-object (ssname ObjSelection index))
33 )

```



```

34 ; récupère le nom de l'objet
35 (Setq NomObj (vla-get-objectname VLAObj))
36
37 ; récupère les coordonnées de tous les sommets sous la forme d'une liste
38 ; (x y xy xy xy ...) ou (x y z x y z x y z ...) selon le type de polyligne
39 (setq PTDepart (vlax-safearray->list
40   (vlax-variant-value
41     (vla-get-Coordinates VLAObj)
42   )
43 )
44 )
45 ; Incrémente un index pour passer un à un tous les sommets
46 (setq index1 0)
47
48 ; Teste le type d'objet polyligne
49 ; si c'est une polyligne optimisée "AcDbPolyline" alors dans ses coordonnées ne figure pas le Z
50 ; si c'est une polyligne non optimisée "AcDB2dPolyline" alors ses coordonnées sont X Y et Z
51 (if (= (vla-get-objectname VLAObj) "AcDbPolyline")
52   (setq ValCoord 2)
53   (setq ValCoord 3)
54 )
55
56 (setvar "clayer" (vla-get-layer VLAObj))
57
58 ; boucle sur la longueur des la liste des sommets
59 ; cette longueur est divisée par 2 ou par 3 selon le type de polyligne
60 (repeat (/ (length PTDepart) ValCoord)
61
62   ; passe dans le sous-programme pour écrire la coordonnée X
63   (EcritureCoordonnees_x
64     (list (nth index1 PTDepart)
65       (nth (1+ index1) PTDepart)
66     )
67   )
68 ; passe dans le sous-programme pour écrire la coordonnée Y
69 (EcritureCoordonnees_y
70   (list (nth index1 PTDepart)
71     (nth (1+ index1) PTDepart)
72   )
73 )
74 ; incrémente le compteur des sommets
75 (setq index1 (+ index1 ValCoord))
76 )
77
78
79 ; on incrémente le compteur des objets dans le jeu de sélection
80 (setq index (1+ index))
81
82 ) ; fin de la boucle Repeat
83 ) ; fin du Progn
84 ) ; Fin du If
85

```

```

86 (setvar "osmode" osmode)
87
88 (setvar "Clayer" CalqueCourant)
89
90 (princ)
91
92 ) ; fin du Defun du programme principal
93
94 ; ===== SOUS-PROGRAMMES =====
95
96 (defun EcritureCoordonnees_X (XY)
97
98   (if (= HauteurStyleCourant 0)
99     (command "_text"
100       xy
101       HauteurDuTexte
102       90
103       (rtos (car XY) 2 (getvar "luprec")))
104   )
105   (command "_text" xy 90 (rtos (car XY) 2 (getvar "luprec")))
106   )
107   )
108
109 (defun EcritureCoordonnees_Y (XY)
110
111   (if (= HauteurStyleCourant 0)
112     (command "_text" xy HauteurDuTexte 0 (rtos (cadr XY) 2 (getvar "luprec")))
113     (command "_text" xy 0 (rtos (cadr XY) 2 (getvar "luprec")))
114   )
115   )
116
117 (defun HauteurTexte ()
118
119   ; récupère le nom du style de texte courant
120   (setq NomStyleTexteCourant (getvar "TextStyle"))
121   ; récupère la hauteur du style de texte courant
122   (setq HauteurStyleCourant (cdr (assoc 40 (tblsearch "Style" NomStyleTexteCourant))))
123   ; texte la hauteur du style de texte courant
124   ; si la hauteur est égale = 0 alors on demande à l'utilisateur d'entrer une hauteur
125   (if (= HauteurStyleCourant 0)
126     (progn
127       ; la valeur est obligatoire et non négative
128       (initget 7)
129       (setq HauteurDuTexte (getreal "\nHauteur du texte : "))
130     )
131     ; sinon on récupère la hauteur définit dans le style de texte courant
132     (setq HauteurDuTexte HauteurStyleCourant)
133   )
134   )

```

Analysons quelque peu ce programme.

Dès que vous créez des objets, pensez à désactiver les accrochages aux objets afin d'éviter d'obtenir des formes totalement aléatoires. La ligne 7 mémorise la valeur des accrochages actuels. La ligne 9 désactive l'accrochage aux objets en mettant une valeur 0 à la variable système "Osmode". A la ligne 86 l'accrochage aux objets reprend sa valeur initiale.

Si on est amené à changer de calque, pensez à mémoriser le nom du calque courant (ligne 11) pour pouvoir le remettre comme il était à la fin du programme (ligne 88).

La ligne 14 exécute le sous-programme "HauteurTexte" de la ligne 117. Ce sous-programme récupère la hauteur de texte défini dans le style de texte courant. Si cette hauteur est égale à zéro, il est demandé à l'utilisateur d'entrer une hauteur puis celle-ci est mémorisée dans la variable "HauteurDuTexte". Si la hauteur est plus grande que zéro, on mémorise simplement la hauteur trouvée. A la fin du sous-programme on retourne sous la ligne qui l'a lancée, c'est-à-dire à la ligne 15.

Les lignes 16 à 22 demandent à l'utilisateur de sélectionner des objets. La sélection sera filtrée pour ne prendre en compte que des objets de type "Polyligne".

La ligne 24 teste si la sélection contient bien des objets.

La ligne 29 crée une boucle de répétition sur le nombre d'objets dans la sélection. Le premier objet est indexé à la valeur 0.

Les lignes 31 à 33 récupèrent les propriétés de l'objet indexé.

La ligne 35 récupère le type de polyligne. Une polyligne non optimisée possède une coordonnée Z tandis qu'une polyligne optimisée n'en possède pas. D'où l'importance de connaître le type de la polyligne car pour obtenir le nombre de sommets il faudra diviser par 2 ou par 3 le nombre de valeur contenu dans la liste des points.

```
(1500.0 900.0 2000.0 900.0 2266.13 1096.55 2607.95 1096.55 2607.95 784.99 3148.89 784.99)
```

Le nombre de valeur dans la liste divisé par 2 donne 6 sommets

```
(1500.0 900.0 0 2000.0 900.0 0 2266.13 1096.55 0 2607.95 1096.55 0 2607.95 784.99 0 3148.89 784.99 0)
```

Le nombre de valeur dans la liste divisé par 3 donne 6 sommets

Les lignes 39 à 44 récupèrent les coordonnées de tous les sommets de la polyligne sous la forme d'une liste de points.

La ligne 60 crée une boucle de répétition sur le nombre de sommet.

Les lignes 63 à 67 exécutent le sous-programme "EcritureCoordonnees\_X" (ligne 109) avec un paramètre "XY". La variable "XY" contient les coordonnées du sommet. Dans ce sous-programme on exécute la commande "TEXTE" pour écrire la valeur "X" du point.

Avant d'écrire la coordonnée on teste si la hauteur du texte, correspondant au style de texte courant, est égale ou plus grande que zéro. Si la hauteur est égale à zéro, on doit indiquer la valeur de la hauteur, sinon il faudra l'ignorer.

Les lignes 69 à 73 exécutent le sous-programme "EcritureCoordonnees\_Y" (ligne 96) selon le même principe que le sous-programme "EcritureCoordonnees\_X" mais pour la coordonnée "Y".

A la ligne 75 on n'oublie pas d'incrémenter le compteur de sommets. Idem à la ligne 80 pour le nombre d'objets contenu dans la sélection.

Le programme principal prend fin à la ligne 92.

Comme vous pouvez le constater, un fichier LSP peut contenir plusieurs fonctions "DEFUN" en tant que commande directe ou indirecte. Lorsqu'on crée ce type de fichier, il faut faire attention aux variables. Soit elles seront déclarées globales et elles pourront être utilisées dans tous les programmes chargés en mémoire dans le document DWG courant, soit les variables seront déclarées en mode local et elles n'agiront que dans le module "DEFUN" où elles auront été définies.

## Conclusion

Dans ce numéro des Cahiers nous avons voulu vous faire découvrir le langage AutoLISP et vous démontrez qu'avec quelques petites lignes de codes on pouvait créer ou manipuler des objets AutoCAD. Nous espérons que ces exemples vous seront profitables pour créer vos propres commandes. N'hésitez pas à piocher des parties de codes. N'hésitez pas à utiliser l'aide en ligne d'AutoCAD même si la documentation est en anglais, les exemples sont assez explicites.

Il y a peu d'ouvrage en français traitant ce langage, mais vous pouvez consulter Internet qui contient beaucoup d'exemples.

## Conseils

Un dernier point important. Si vous souhaitez vous lancer dans ce langage, commencez par des petits programmes et testez vos lignes de codes au fur et à mesure. Pour bien mémoriser les fonctions, évitez de trop utiliser les Copier/Coller, forcez-vous à taper vos lignes de code.

# Les Cahiers d'AutoCAD existent aussi pour AutoCAD LT

[www.dominique-vaquand.com](http://www.dominique-vaquand.com)

## BULLETIN D'ABONNEMENT

Bulletin d'abonnement à retourner avec votre règlement aux :

Dominique VAQUAND Informatique  
24 Rue des Icards  
BP 33  
13430 EYGUIERES - FRANCE -

Nom et Prénom .....  
Société .....  
Adresse .....  
Code Postal ..... Ville .....  
Pays ..... Tél ..... Fax .....  
Adresse Email .....

Ci-joint mon règlement de ..... € TTC (Une facture acquittée est systématiquement adressée)

☐ ABONNEMENT POUR 4 N° À LA REVUE «LES CAHIERS D'AUTOCAD»  
40 € TTC (TVA 19.6 % incluse)

A PARTIR DU NUMÉRO : .....

☐ Commande au numéro  
12 € TTC (TVA 19.6 % incluse)

Le(s) numéro(s) : .....