# Programmeren in AutoLISP

## Les drie

## Vormen van data in AutoLISP

Functies en variabelen

door

Joop F. Moelee

een gelovig volger van
“de Sacrale Kunst van Luiheid”
zijn Hoge Priester LISP en Acoliet Script

# 1) Inhoudsopgave

# 2) Verantwoording

Dit document is gebaseerd op en/of bevat delen van de artikelen geschreven door Kenny Ramage eigenaar/ beheerder van de site http://www.afraLISP.com/
Kenny heeft de schrijver dezes toestemming gegeven tot gebruik van zijn teksten.
Hiervoor heel hartelijk bedankt, Kenny.

Voor alle duidelijkheid eerst even de verklaringen van de diverse tekstsoorten.
*Italic--Engelse woorden en termen die geen AutoLISP en/of AutoCAD termen zijn.*
***Italic en vet – AutoCAD termen, commando's en menu opdrachten.***
**Kleiner en vet – regels programmacode en AutoLISP termen.**

# 3) Data types

AutoLISP herkent diverse vormen van data:

- Functies (subroutines, **subrs**)
  - \+ optellen
  - **-** aftrekken
  - ***** vermenigvuldigen
  - \ delen
  - **command** \
  - **princ** |> AutoLISP instructies
  - **setq** /

- Functie argumenten:
  - variabelen, deze staan ook bekend als symbolen (**symbols**)
  - constanten, een heel of een decimaal getal (**integer, real**)
  - tekst, (**character strings**)
  - bestands namen, (**descriptors**)
  - AutoCAD entiteit namen
  - AutoCAD selectie sets
  - externe functies, deze worden aangeleverd door het *AutoCAD Development System*

# 4) Variabelen

De variabelen binnen AutoLISP zijn verdeeld in twee hoofdgroepen, te weten globale (**global**) variabelen en lokale (**local**) variabelen. In sommige documenten wordt de globale variabele ook wel wereld variabele genoemd. (global = wereldwijd).

Het verschil tussen deze twee variabelen is eenvoudig: een routine of programma kan alleen globale variabelen meenemen naar een andere routine en/of programma.
Ter illustratie het volgende programma.

1. Kopieer de code naar **VisualLISP Editor Window**.
2. Stop alle drie de variabelen in de **Watch Window**.
3. Plaats de cursor meteen voor het haakje openen van de eerste setq functie en druk op F9. Er verschijnt nu een rood blokje over het haakje: dit is een breekpunt.
4. Laad het programma in de Console Window en start het programma.
5. Gebruik F8 om stap voor stap door het programma te wandelen.
6. Observeer wat de variabelen doen en wat er wordt afgedrukt.

```
;;;                                                    ;
;;;    Testen van gedrag globale en lokale variabelen  ;
;;;                                                    ;

(defun FunctieA (/ LokaleVarA LokaleVarB)
  (setq LokaleVarA "boo")
  (setq LokaleVarB "hoo")
  (setq GlobaleVarA "foo")
  (FunctieB)
  (print LokaleVarA)
  (print GlobaleVarA)
)

(defun FunctieB (/ LokaleVarA LokaleVarB)
  (print LokaleVarA)
  (print LokaleVarB)
  (print GlobaleVarA)
  (setq GlobaleVarA "Wat je wilt")
  (setq LokaleVarA "fie")
  (functieC)
)

(defun FunctieC (/ LokaleVarA LokaleVarB)
  (print LokaleVarA)
  (print GlobaleVarA)
  (setq GlobaleVarA "Wanneer je wilt")
)
```

Het blijkt  dat alleen de waarde van de globale variabelen meegenomen kunnen worden naar andere programma's en/of routines.
Merk ook op dat de lokale variabelen tussen de haakjes achter de defun opdracht staan:

**(defun FunctieA (/ LokaleVarA LokaleVarB)**

en de globale variabelen niet.
Lokale variabelen worden na afloop van het programma uit het geheugen gewist. Dit staat bekend als A*utomatic Garbage Collection.*
Werk zo veel mogelijk met lokale variabelen, dit maakt het *debuggen* een stuk eenvoudiger en bovendien wordt dan het geheugen van de computer minder vervuild. Deze vervuiling gaat ten koste van de snelheid. Vooral bij grotere en ingewikkelde programma's is dit merkbaar.

Maar wat doet die schuine streep daar nu in **(/ LokaleVarA LokaleVarB)** ?
Alle variabelen die voor de schuine streep staan zijn waarden die aan het commando/opdracht meegegeven worden van buiten af, dus door de gebruiker en/of de routine die het commando start.

# 5) Debuggen

Zoals je in de *Watch window* kunt zien staat na beeindiging van het programma de lokale variabelen op ***nil.*** Dus ze hebben geen waarde. Dit is erg onhandig tijdens de ontwikkeling van een programma. Het is daarom handiger de variabelen als **global** te definiëren totdat het programma volledig is getest en pas nadat je tevreden bent over de werking de **local** variabelen te kopiëren naar de haakjes achter de functienaam.

# 6) Tot slot

Ik hoop dat nu een beetje duidelijker is geworden wat variabelen zijn en welke er bestaan. In de loop van deze *tutorial* zullen we hier nog enkele keren op terug komen.
LISP is ontwikkeld om het manipuleren van lijsten te vergemakkelijken.
Daarom gaan we het in les vier hebben over lijsten en wat er mee gedaan kan worden.

Tot dan,

Joop