LOGO

aanvullende cursus horende bij de LOGO-lessen gegeven door Steven Stinis

Er wordt verwacht dat je:

- de eigenschappen van LOGO inziet
- de commander, editor en menu's kunt gebruiken
- het nut van procedure's, variabelen, parameters, abstractie en recursie inziet en zinvol kan toepassen

De nadruk ligt op kunnen, niet op kennen. Alle toetsen zijn dus open boek. Bij sommige vragen mag je MSWLogo gebruiken. De ideale voorbereiding op een toets is dus met LOGO werken, veel oefeningen maken en jezelf vragen stellen bij wat je doet. Wie aandachtig de les volgt en zelf wat experimenteert met LOGO mag hiermee geen problemen hebben.

De leerstof omvat de slides die in de les gezien zijn. Deze cursus dient als ondersteuning, het is dus zeker geen extra leerstof.

INHOUD

HOOFDSTUK 1 : Welkom bij Logo

1.1 De kenmerken van LOGO

a) LOGO is interactief

Dit wil zeggen dat het met LOGO mogelijk is iets te proberen, en daarvan direct het resultaat op het scherm te zien. LOGO wordt dus geïnterpreteerd en niet vertaald!

b) LOGO is procedureel

Een groot probleem kan als het ware opgesplitst worden in vele kleintjes. Dit vergemakkelijkt het proces van verfijning. Een deelalgoritme kan dus geïmplementeerd worden als een aparte procedure.

c) LOGO is uitbreidbaar

LOGO biedt een gamma van opdrachten en bewerkingen aan (“*primitieven*”). Maar het is voor iedere gebruiker mogelijk zijn eigen primitieven bij te maken.

d) LOGO beschikt over de mogelijkheid dat procedures zichzelf aanroepen (recursie)

Recursie is een zeer krachtig aspect van LOGO met zeer veel gebruiksmogelijkheden.
Een procedure roept dus zichzelf steeds opnieuw aan en dit kan, als we niet ingrijpen, eeuwig zo doorgaan. Het bijzondere schuilt in het feit dat de voorbeeldprocedure steeds een iets andere uitwerking heeft. Op die manier kunnen we tot prachtige resultaten komen.

1.2 De vier LOGO-werelden

LOGO is opgesplitst in een aantal leeromgevingen, ook wel LOGO-werelden genoemd. De vier LOGO-werelden laten elk een ander aspect van LOGO zien. Ze kunnen elk afzonderlijk, in samenhang of tegelijkertijd aangesproken worden.

We onderscheiden :
- De *schildpadwereld* : de wereld van tekenen en rekenen.
- De *taalwereld* : manipuleren met woorden, lijsten en variabelen.
- De *muziekwereld*.
- De *sprokenwereld* : de wereld van animaties en bewegende beelden.

HOOFDSTUK 2 : De LOGO-schildpadwereld

2.1 Enkele basisprimitieven

De wereld van de Turtle Graphics is zeer fascinerend. We komen terecht in een wereld van lijen, vormen en kleuren. Een wereld die zich kenmerkt door de "Turtle Geometry" (schildpad-georiënteerde meetkunde).

In de les maken we gebruik van *MSWLogo*, bij deze versie verschijnt initieel een driehoekje centraal in de werkruimte, dit stelt de *schildpad* voor.

Om de schildpad te doen verdwijnen en om hem opnieuw op het scherm te toveren maken we gebruik van volgende instructies :

```
ST          (show turtle)
HT          (hide turtle)
```

Om te tekenen, zullen we de schildpad in beweging moeten zetten.

```
FD getal    (forward + aantal stappen)
BK getal    (back + aantal stappen)
HOME        (breng schildpad terug naar uitgangspositie,
            midden    op het beeld met zijn neus verticaal
            naar boven, de schildpad laat hierbij een spoor
            na)
```

Bij de opdracht `FD 100` loopt de schildpad dus 100 stappen vooruit, vanuit de uitgangspositie is dit dus omhoog.

Opmerking :
Na **`FD`** *of* **`BK`** *moet er minstens één spatie en een getal staan!*

We kunnen de schildpad ook laten draaien met volgende commando's :

```
RT getal    (right + aantal graden)
LT getal    (left + aantal graden)
```

Bij `RT 90` draait de schildpad over 90° naar rechts, bij `LT 270` draait hij 270° naar links, hetgeen hetzelfde resultaat geeft!

Opmerking :
Ook hier dient na **`RT`** *of* **`BK`** *minstens één spatie en een getal te staan*.

Om het scherm weer schoon te maken gebruiken we :

```
CS          (clear screen, voor het tekengedeelte)
CT          (clear text, voor het tekstgedeelte)
```

Met de vier opdrachten `FD`, `BK`, `RT` en `LT` is bijna elke denkbare tekening te maken.

Als voorbeeld gaan we een vierkant maken :

Eerst laten we de schildpad 100 stappen vooruit lopen (verticaal naar boven). Daarna draaien we hem 90° rechtsom. De looprichting is nu horizontaal naar rechts. We lopen nu weer 100 passen vooruit om vervolgens weer 90° naar rechts te draaien. Dit herhalen we nog twee keer. Uiteindelijk krijgen we :

```
FD 100 RT 90 FD 100 RT 90 FD 100 RT 90 FD 100 RT 90
```

Zo ontstaat een keurig vierkant.

Deze manier van het gebruiken van LOGO-opdrachten is nogal omslachtig. Het blijkt dat we in het bovenstaande voorbeeld twee opdrachten vier maal moeten herhalen. De volgende instructie biedt enig soelaas :

REPEAT *getal* [*opdracht*]

Opmerking :

***REPEAT** moet altijd gevolgd worden door minstens één spatie, een getal en de opdrachten die herhaald moeten worden. Deze opdrachten (het kan ook een procedure zijn) moeten altijd tussen vierkante haakjes geschreven worden.*

Als we even terugkeren naar het vierkant :

```
REPEAT 4 [FD 100 RT 90]
```

2.2 Procedures maken

Tot nu toe hebben we in LOGO steeds gewerkt in de "*Direct Mode*". Dat wil zeggen dat elke opdracht na het indrukken van de enter-toets direct werd uitgevoerd. Maar LOGO biedt de gebruiker ook de mogelijkheid om programma's te maken, we noemen hier een programma een *procedure* (LOGO is proceduraal!).
Procedures maken doen we in de Editor. Deze roepen we op met

EDIT "*procedurenaam*

In LOGO heeft iedere procedure dus een eigen naam!
Elke procedure heeft END als laatste opdracht.
We verlaten de editor via File-Save and Exit.

Als voorbeeld schrijven we een procedure om een vierkant te tekenen :

Toetsen we

EDIT "VIERKANT

Dan kunnen we in de editor tussen de regels to VIERKANT *en* end *de gewenste lijn(en) voegen, we krijgen :*

```
to VIERKANT
REPEAT 4 [FD 100 RT 90]
end
```

De gemaakte procedures worden door LOGO opgeslaan in het werkgeheugen. Om nu de procedure te laten uitvoeren hoeven we hem alleen maar aan te roepen door het intoetsen van de procedurenaam, gevolgd door een enter.

We kunnen ze echter ook door een andere procedure laten aanroepen, we spreken dan van een *subprocedure*, bvb. :

```
EDIT "STER

to STER
REPEAT 36 [VIERKANT RT 10]
end
```

Als we de procedure ster aanroepen, tekent de schildpad een vierkant (op voorwaarde dat we de procedure vierkant gedefinieerd hebben) en draait dan over 10 graden rechtsom. Dan tekent hij nog een vierkant en draait weer 10 graden rechtsom…
Tenslotte komt de schildpad terug in zijn uitgangspositie, want 36 keer 10 graden is 360 graden, een volle hoek.

Oefeningen :
- Schrijf een procedure om een regelmatige driehoek te tekenen.
- Schrijf een procedure om een regelmatige 36 hoek te tekenen, wat merk je op als we de zijden niet al te groot nemen?
- Schrijf een procedure om een rechthoekige driehoek met rechthoekszijden lengte 200 te tekenen

---

2.3 Variabelen

De procedure vierkant is aardig, maar nogal beperkt in zijn mogelijkheden doordat we slechts in staat zijn vierkanten te teken met zijde 100 stappen. Om met dezelfde procedure verschillende vierkanten te kunnen tekenen moeten we het aantal stappen *variabel* maken, dat kunnen we op volgende manier :

```
EDIT "VIERKANT :ZIJDE

to VIERKANT :ZIJDE
REPEAT 4 [FD :ZIJDE RT 90]
end
```

Achter de procedurenaam hebben we nu een variabele gezet die zijde heet. Om LOGO te laten zien dat het een variabele en geen procedure is, zetten we er een dubbele punt voor.
Deze variabel wordt gebruikt om de schildpad te vertellen hoeveel stappen hij vooruit moet gaan. Door in te toetsen `VIERKANT 88` geven we de variabele `:ZIJDE` de waarde 88.
Op deze manier kunnen we elk gewenst vierkant tekenen. Geven we per ongeluk alleen `VIERKANT` in, dus zonder getal, dan geeft LOGO volgende foutboodschap :

```
NOT ENOUGH INPUTS TO VIERKANT
```

Als we een procedure als subprocedure willen gebruiken moeten we de eventueel daarin gebruikte variabelen ook bij de hoofdprocedure vermelden, bvb. :

```
EDIT "STER :LENGTE

to STER :LENGTE
REPEAT 36 [VIERKANT :LENGTE RT 10]
end
```

Het is ook mogelijk twee of meerdere variabelen te gebruiken, bvb. :

```
EDIT “VEELHOEK :LENGTE :HOEKEN

to VEELHOEK :LENGTE :HOEKEN
REPEAT :HOEKEN [FD :LENGTE RT 360/:HOEKEN]
end

VEELHOEK 100 4   geeft ons een vierkant met zijde 100,
VEELHOEK 50 3   een regelmatige driehoek met zijde 50.
```

We kunnen de variabelen ook op een andere manier een waarde geven. Dat doen we dan met MAKE :

```
MAKE “LENGTE 100
```

Hiermee maken we een variabele die we lengte noemen, we kunnen die variabele uitschrijven aan de hand van :

```
PRINT :LENGTE
```

Let er op dat we om de variabele aan te maken gebruik maakten van de dubbele aanhalingstekens, en in verder gebruik ervan terug het dubbele punt gebruiken!

We hebben de variabele lengte de waarde 100 gegeven, die kunnen we evenwel nog veranderen met volgend commando :

```
MAKE “LENGTE :LENGTE + 100
```

Hierdoor vermeerderen we de waarde van de variabele met 100.

Een eenvoudig voorbeeld :

```
EDIT “BLOKKENDOOS :ZIJDE :KEER

to BLOKKENDOOS :ZIJDE :KEER
REPEAT :KEER [VIERKANT :ZIJDE MAKE “STAP :STAP + 10]
end

BLOKKENDOOS 20 5 zal 5 keer een vierkant tekenen, het eerste vierkant met zijden van 20
stappen, het tweede met zijden van 30 stappen, …
```

2.4 Recursie

Recursie is een proces dat verwijst naar zichzelf. Het is ook mogelijk dat een procedure zichzelf aanroept terwijl hij zichzelf aanroept. Dit kan eindeloos doorgaan. We kunnen het vergelijken met iemand die zichzelf bekijkt in een spiegel die voor hem staat, en een spiegel die achter hem staat.

Bekijken we volgend voorbeeld :

```
EDIT “VIERKANT :ZIJDE

to VIERKANT :ZIJDE
REPEAT 4 [FD 100 RT 90]
VIERKANT :ZIJDE
end
```

Deze procedure blijft zichzelf aanroepen en zal dus niet stoppen. Om dit op te lossen drukken we op de HALT-toets in de command-box (MSWLogo), in sommige versies van LOGO volstaat een druk op de break-toets.

Er zijn verschillende betere manieren om een recursieve procedure te stoppen.

a) Een eerste manier is om in de procedure een *testopdracht* te plaatsen. Dit kan door gebruik te maken van de keuzeopdracht, die in de meeste programmeertalen gedefinieerd is :

```
IF voorwaarde [opdracht]
```

Als aan een bepaalde voorwaarde voldaan is, voer opdracht uit.

```
IFELSE voorwaarde [opdracht1] [opdracht2]
```

Als aan een bepaalde voorwaarde voldaan is, voer opdracht1 uit, zoniet, voer opdracht 2 uit.

Een andere nuttige primitieve in dit verband :

```
STOP
```

Hiermee maak je een einde aan de procedure.

We bekijken een eenvoudig voorbeeld :

```
EDIT “VIERKANT :ZIJDE

to VIERKANT :ZIJDE
IF :ZIJDE  > 150 [STOP]
REPEAT 4 [FD :ZIJDE RT 90]
VIERKANT :ZIJDE + 5
end
```

Dit is een eenvoudige maar prima recursieve procedure!
Deze procedure is qua uitwerking te vergelijken met procedure blokkendoos. Er wordt eerst een vierkant getekend met zijden gelijk aan de variabele, vervolgens

een vierkant met zijden 5 stappen dan die van het eerste vierkant, … de procedure stopt wanneer de variabele groter wordt dan 150.
Als opmerking kunnen we nog formuleren dat we ook hadden kunnen gebruik maken van de opdracht `MAKE "ZIJDE :ZIJDE + 5` *om de variabele te veranderen en daaronder* `VIERKANT :ZIJDE ,` *om de variabele te veranderen.*

b) Een andere manier om een recursieve procedure te laten stoppen is het inbouwen van een *teller*, die bijhoudt hoe vaak de procedure zichzelf herhaalt. Bvb. :

```
EDIT "FIGUUR :VOOR :HOEK :TELLER

to FIGUUR :VOOR :HOEK :TELLER
IF :TELLER < 1 [STOP]
FD :VOOR RT :HOEK
MAKE "VOOR :VOOR + 10
FIGUUR :VOOR :HOEK :TELLER – 1
end
```

2.5 LOGO pen- en kleuropdrachten

De schildpad beschikt over een tekenpen, waarmee hij zijn spoor op het scherm kan trekken. Er zijn echter functies aanwezig waarmee het spoor van de schildpad verandert :

```
PU          (pen up, vanaf nu is het spoor onzichtbaar)
PD          (pen down, vanaf nu is het spoor weer zichtbaar)
PE          (pen erase, de tekenpen wordt een vlakgum)
PX          (pen reverse, de pen tekent waar niks staat, en
                    gomt waar wel iets staat)
```

Voorbeelden :

```
PD FD 100 WAIT 50 PE BK 100
```

Er wordt een lijn getekend, die enkele tijdseenheden later weer wordt gewist.

```
EDIT "GROEIVIERKANT :S

to GROEIVIERKANT :S
IF :S > 150 [STOP]
PX REPEAT 2 [VIERKANT :S]
GROEIVIERKANT :S + 5
end
```

Doordat per grootte het vierkant eerst getekend wordt en vervolgens gewist wordt krijgen we de indruk dat het vierkant groeit.

Oefening :

Schrijf een procedure die een vierkant laat draaien om zijn eigen as.

LOGO is ook een wereld van kleuren, het is mogelijk

a) de kleur van het *scherm* aan te passen.
b) de kleur van de *pen* aan te passen.
c) *oppervlakken* op te vullen met een bepaalde kleur.

In MSWLogo werken we met een kleurenpalet dat bestaat uit 3 kleuren : rood, groen en blauw. Door tinten gelegen tussen 0 en 255 voor iedere kleur op te geven kunnen we 16,7 miljoen verschillende kleuren genereren.

De algemene vorm van een *kleurenpalet* is :

```
[rood groen blauw]
```

Bvb. :

```
[0 0 0]           zwart
[255 255 255]     wit
[128 128 128]     grijs
[255 0 0]         rood
[0 255 0]         groen
[0 0 255]         blauw
```

a) De kleur van het scherm

```
SETSC kleurenpalet      (pas schermkleur aan, ook setscreencolor
                        en setscreencolour werken!)
```

Opmerking:
Pas eerst de schermkleur aan voor dat je begint te tekenen!

Met de functie `SHOW SCREENCOLOR` is het mogelijk de actuele schermkleur (palet) op te vragen.

b) De kleur van de pen

```
SETPC kleurenpalet      (SETPENCOLOR/SETPENCOLOUR)
```

Met de functie `SHOW PENCOLOR` is het mogelijk de actuele penkleur op te vragen.
Het is eveneens mogelijk de dikte van de pen aan te passen :

SETPENSIZE [breedte hoogte]

c) Oppervlakken opvullen

```
FILL        (vul het oppervlak waarin zich de schildpad bevindt)
```

De opvulkleur wordt ingesteld aan de hand van :

```
SETFC kleurenpalet            (setfloodcolor/setfloodcolour)
```

Met de functie `SHOW FLOODCOLOR` is het mogelijk de actuele opvulkleur op te vragen.

Voorbeeld :

```
repeat 4 [fd 100 rt 90]
rt 45
pu
fd 20
fill
```

We kunnen ook aan de hand van

```
SETX getal
SETY getal
SETXY getal getal
SETPOS [getal getal]
```

de positie van de schildpad veranderen, hierbij laat de schildpad een spoor na.
Met de functie `show pos` krijgen we actuele positie op het scherm.