Go: een snelle introductie

Auteur: Daniël Heres
Datum: Januari 2010

3,2,1 Go!

"Weer een nieuwe taal!" zul je wel denken. De keuze in programmeertalen is groot en de gelijkenis vaak ook. Toch is er een ding dat vaak overeind blijft: het gebruik van talen met statische types (system language). Deze talen zijn een stuk sneller in het uitvoeren van algebra omdat er bijvoorbeeld met 32-bits typen gerekend kan worden terwijl bij de meeste dynamische talen er met 64-bit floats gerekend wordt. Een ander groot verschil tussen dynamische en statische talen is vaak de complexiteit van een taal: talen als C++ kennen heel veel sleutelwoorden die in verschillende volgorden in de code kunnen voorkomen terwijl een taal als JavaScript (dialect van ECMAScript**)** maar één mogelijkheid kent. Dit maakt het voor C++ wel mogelijk om meer optimalisaties in te bouwen, maar eenvoudigere talen kunnen (vooral wanneer het aantal bestanden toeneemt) veel sneller gecompileerd worden. Dat blijkt vooral handig handig bij het ontwikkelen van programma's en bij JIT-(Just In Time) compileren. Go is een taal niet echt als doel heeft een nieuwe syntax door te voeren (zoals veel nieuwe dynamische talen) maar de snelheid van talen met statische types te combineren met het gemak van dynamische talen en een supersnelle compilatie. Deze taal wordt ontwikkeld door Google. De drijfveer van Google is duidelijk: webapplicaties maken met veel meer mogelijkheden die op elk platform kunnen draaien en daarnaast snel afgeleverd worden. Go maakt een goede indruk als een alternatief voor system languages en voor JavaScript.

Deze gids is gemaakt voor mensen die al wat programmeerervaring hebben en snel basisconcepten van deze nieuwe taal willen leren.

Commentaar

Commentaar is hetzelfde als veel andere talen:

```
/* Dit is commentaar
over meerdere
regels */
// Commentaar voor een regel
```

Pakketten

De eerste regel van Go moet een pakketnaam beschrijven. In een Go project moet er minstens één pakket zijn met de naam main en die moet een functie met de naam main bevatten.

```
package kabouters // Andere pakketten in project kunnen dit kabouterpakket importeren

import "fmt" // Pakket bekend door compiler

import "./mutsen" // Pakket beschikbaar in map
```

Variabelen en constanten

Het declareren van variabelen en typen gaat in Go behoorlijk anders dan in de meeste talen. Typen worden achteraf gedeclareerd, na de namen. Dit is duidelijk te zien in het volgende voorbeeld:

```
var a int // Variabele a heeft type int. Let op: een puntkomma als einde voor een statement is voor Go niet noodzakelijk (mag wel)
```

```
var b, c, d, e int32 = 5, 7, 10, 4  // b, c, d en e hebben allemaal type int32, een regel mogen meerdere variabelen declareren. Let op int32 is niet hetzelfde als int

const r int = 5 // e is een constante

var f = 5 // Dit kan bij numerieke types, types worden automatisch toegewezen

var g string = "Go go go!"

var (
    h int;
    i string;
    j int32 = 45;
    k, l, m = "GOOG", 43, 1.0
) // Gebruik van haakjes is heel handig bij declareren van veel variabelen, puntkomma's zijn verplicht als scheidingsteken! Haakjes kunnen ook bij sleutelwoorden const en type gebruikt worden.

const (
  Maandag, Dinsdag, Woensdag = 1, 2, 3;
  Donderdag, Vrijdag, Zaterdag = 4, 5, 6;
)



//Let op, dit mag alléén binnen functies!
func functies() {
    n := 4 // Dit is een speciale korte declaratie voor variabelen

    n := functie0() // Zeer comfortabel bij het aanroepen van functies!

    o, p, q := 5.7, 100, "Kort"
}
```

Merk op dat er aandacht besteed is aan het zo kort mogelijk opschrijven van meerdere variabelen.
De laatste constanten kunnen bijvoorbeeld door een programma verkleind worden tot

```
const(a,b,c,d,e,f=1,2,3,4,5,6)
```

Dat is natuurlijk geen toeval. Grote programma's moeten zo snel mogelijk over een netwerk verstuurd worden, daar helpt een korte notatie natuurlijk goed bij! Hier zal ook bij de volgende concepten goed over nagedacht zijn!

Tot slot, de numerieke types zijn:

int	uint	float
int8	uint8 of byte	
int16	uint16	
int32	uint32	float32
int64	uint64	float64

Assignments

Laten we meteen maar met een paar voorbeelden beginnen!

a = x

b, c = functie1(), 5 // Meerdere assignments, we zien in het volgende waar dat heel handig voor is

d, e = e, d // Wisselen van variabelen! Hiervoor zijn bij minder expressieve talen drie regels voor nodig

If

Voor een if-statement worden geen haakjes gebruikt. De rest is gelijk aan de meeste andere talen.

```
import "fmt" // importeert pakket fmt, meer over dat later

if x<a { klein() }
else if x==a { gelijk() }
else x>a { groter() }

if v:=s(); v<10 { // v wordt gedeclareerd, let op de puntkomma!
    fmt.Printf("Go " + v) // Geeft Go plus uitvoer van s in console weer
}
```

Operators

Voorrang	
6	/ % << >> & &^
5	+ - | ^
4	== != < <= > >=
3	<-
2	&&
1	||

For

Een for loop gebruikt ook geen haakjes. Een for loop heeft twee verschillende modussen: één met één element en één met drie elementen.

for a<5 {} // Dit Staat gelijk aan een while statement in andere talen

for {} // While(true)

for a:=0;; a++{} //While(true) en een counter!

for a:=0; a<10; a++ {} // Een gewone for-loop

for i := range x {} // Probeer deze op verschillende types! Let op: werkt niet voor

integers zoals bijvoorbeeld bij programmeertaal Python.

Switch

Switches zijn grotendeels gelijk aan andere talen met een paar belangrijke verschillen: alle types kunnen gebruikt worden en er is geen break nodig (waarmee vaak fouten worden gemaakt), het switch statement stopt standaard zodra er een kloppende voorwaarde is.

```
switch f(10){
   case 100: Print("Vertienvoudigd!")
   case 10: Print("Gelijk!")
   case 2,3,5: Print("Nu, dit is wezenlijk verrassend!")
   default: Print("Standaard!")
}

switch f:=f(10);{ // Je kan ook meerdere variabelen of declareren!
   case f>100: Print("Erg groot");// Expressies in een switch statement
   case f>10: Print("Aardig groot") fallthrough; // Gaat door naar volgende case, ook als hij waar is
   case f==20: Print("Aardig groot en ook nog 20!")
}
```

Functies

Kenmerkend voor Go zijn de notatie van parameters en de mogelijkheid om meerdere waardes te "returnen". Een functie wordt gedeclareerd met "func".

```
func Wortel(parameter int) (float, bool) {
   if parameter > 0 { math.Sqrt(parameter), false }
   return 0, false
}

func Konijn(wortels int) (genoeg bool) { // Als je parameters benoemt kun je ze gebruiken
   if wortels > 10 { genoeg = true; }
   return genoeg
}

func Konijn(wortels int) (genoeg bool) {
   if wortels > 10 { genoeg = true; }
   return // genoeg wordt "gereturned", kleiner dan tien -> false
}

func FunctieInFunctie(i int) (int) (
   g := func(j int) (int) {
      h := j * i + 1;
      return h
   }
   return g(i +5);
}
```

Arrays

```
var x = [4]int{2, 8, 19, 30} // Maak een array met lengte 4 en waardes 2, 8, 19 en 30
var y = [4]int{2, 9} // Array met lengte 4, met waardes 2, 9, 0, 0
var z = [100]int{20:7, 65:8,99:2} // Array met lengte 100. Index 20 heeft waarde 7, 65 waar 8 enz. De rest heeft de waarde 0.

var a = [4]string{"bas", "bink", "bobo"} // index 3 is empty string
var b = [7]MijnEigenType{} // Eigen types kunnen ook, volgt later!

func arrays() {

   for i := range x {
      fmt.Print(i) // 0, 1, 2, 3
      fmt.Print("\n")

   for _, j := range x {  // Een array heeft keys (index) en waardes, indien je keys niet nodig hebt gebruik je een laag streepje om alleen de waardes op te vragen!
      fmt.Print(j) // 2, 8, 19, 30
      fmt.Print("\n")
   }

   for i := range x {
      x[i]  = i
   } //Maakt {0, 1, 2, 3}

   x[2] = 0

   fmt.Print(x[1:len(x)]) // Print alles behalve de eerste. len(x) rekent lengthe array uit.

}
```

Slices

Slices zijn vrij uniek voor Go. Slices kan je definiëren als arrays met een variabele lengte.

```
var x = []int{} // integer slice
```