Seminario de Lenguajes opción Go

Raúl Champredonde

Seminario de Lenguajes opción Go

- Programa
- Package
- Variables
- Constante
- Tipo básicos
- Operadores
- Asignación

Qué es Go?

- Lenguaje de programación multiplataforma de código abierto
- Compilado, fuertemente tipado
- Sintaxis basada en C / C++ (al igual que Java, PHP, Phyton, C#, etc.)
- Desarrollado por Google en 2007

Para qué se utiliza Go?

- Desarrollo web (lado del servidor)
- Desarrollo de aplicaciones en red
- Desarrollo de aplicaciones multiplataforma
- Desarrollo nativo de la nube
- Desarrollo de aplicaciones concurrentes

Por qué usar Go?

- Es fácil de aprender
- Tiene tiempos de ejecución y de compilación rápidos
- Admite la concurrencia
- Admite genéricos
- Tiene administración de memoria
- Es portable a diferentes plataformas (Windows, Mac, Linux, etc.)

Estructura de un programa

Package declaration

Import packages

Functions

Statements and expressions

```
package main

import ("fmt")

func main() {
    fmt.Println("Hello World!")
}
```

Estructura de un programa

```go
package main

import ("fmt")

func main() {
   fmt.Println("Hello World!")
}
```

```go
package main

import (
    "fmt"
    "math/rand"
)

func main() {
  fmt.Println("N:", rand.Intn(10))
}
```

```go
import ("fmt")
import ("math/rand")
```

- Un programa es parte de un package. Se define con la palabra clave package y el nombre.
- `import ("fmt")` permite importar las librerías a utilizar en el programa.
- Go ignora las líneas en blanco.
- `func main() {}` es una función. El código incluido en sus {} será ejecutado.
- `fmt.Println()` es una función que está disponible dentro del package fmt. Es usado para imprimir texto.

Consideraciones generales

- Todo ejecutable pertenece al package “main”.
- Cada sentencia se separa con un carácter “fin de línea” (<enter>) o con “;”.
  Ejemplo posible:
  ```
  package main; import ("fmt"); func main() { fmt.Println("Hello World!");}
  ```
- La llave inicial de un bloque (función o estructura de control) no puede estar al inicio de una línea.
- Comentarios:
  - Una línea: `// ...`
  - Varias líneas: `/* ... ... */`

Exportación de identificadores

- Los identificadores declarados en un package son “exportados” (es decir, visibles desde afuera) cuando comienzan con mayúscula.

```
package main
import (
    "fmt"
    "math"
)
func main() {
    fmt.Println(math.Pi)
}
```

- `Println` es una función exportada por el package `fmt`
  `Pi` es una constante exportada por el package `math`

Variables

- Las variables se declaran con la palabra clave `var`:

```
var i int
var c, d, e bool
```

```
var (
    i int
    c, d, e bool
)
```

- La declaración de variables puede incluir su inicialización:

```
var i int = 1
var i, j int = 1, 2
```

- Si hay una inicialización, el tipo puede ser “inferido” de los valores:

```
var c, d, e = true, false, "Texto"
```

- El tipo puede ser “inferido” usando “:= ” (short assignment):

```
k := 3
```

- Los nombres de las variables:
  - Son case-sensitive
  - Pueden contener cualquier cantidad de “a” - “z”, “A” - “Z”, “0” - “9”, “_”, pero no comenzar con “_”.
  - No puede ser igual que una palabra reservada.

Variables

- Se puede declarar variables en un package o en una función:

```
package main
import "fmt"
var b, c, d bool
func main() {
   var i int
   fmt.Println(i, b, c, d)
}
```

- El ":= " sólo se puede usar dentro de funciones ya que en el package toda sentencia debe comenzar con una palabra clave (`var`, `func`, etc.).

Constantes

- Se declaran como las variables pero con la palabra clave `const`.
- Pueden tener valores character, string, boolean, or numéricos.
- Se puede declarar constantes sin tipo:

```
const Pi = 3.14
const A = 1
```

```
const (
    A int = 1
    B = 3.14
)
```

- No se puede declarar constantes con “:= ”.

Tipos básicos

- `bool`: Valores: `true` / `false`. Valor por defecto: `false`
- `string`: Valor por defecto: `""`
- `int`, `int8`, `int16`, `int32`, `int64`, `uint`, `uint8`, `uint16`, `uint32`, `uint64`, `uintptr`: Valor por defecto: `0`. Usar siempre `int` excepto que haya alguna razón específica para usar otro tipo entero. Valor por defecto: `0`
- `byte` (alias de `uint8`)
- `rune` (alias de `int32`)
- `float32`, `float64`: Valor por defecto: `0`
- `complex64`, `complex128`: Valor por defecto: `(0+0i)`

Conversión de Tipos

- Si `T` es un tipo y `v` es un valor,
  la expresión `T(v)` convierte el **valor** de `v` al tipo `T` .
- Se puede aplicar sobre tipos numéricos o si los tipos origen y el destino tienen el mismo tipo subyacente.

```
var i int = 42
var f float64 = float64(i) + 0.000001
var u uint = uint(f)
var g float64 = float64(int(f))

fmt.Println(i, f, u, g)
```

```
i := 42
f := float64(i) + 0.000001
u := uint(f)
g := float64(int(f))

fmt.Println(i, f, u, g)
```

```
// 42
// 42.000001
// 42
// 42
```

Named types

```
type Celsius float64
type Fahrenheit float64
```

- Los tipos nombrados permiten que el compilador controle la mezcla no intencional.
- Tienen la misma estructura y operaciones que el tipo subyacente.
- Se pueden convertir si tienen el mismo tipo subyacente.

```
func CToF(c Celsius) Fahrenheit {
    return Fahrenheit(c * 9 / 5 + 32)
}
```

```
func FToC(f Fahrenheit) Celsius {
    return Celsius((f - 32) * 5 / 9)
}
```

Operadores

- Operadores aritméticos
  - Se aplican sobre tipos numéricos; operandos del mismo tipo

Operador	Descripción	Ejemplo
+	Suma	x + y
-	Resta	x - y
*	Multiplicación	x * y
/	División	x / y
%	Módulo	x % y
-	Menos unario	-x
+	Más unario	**+x**

Operadores

- Operadores sobre strings

Operador	Descripción	Ejemplo
`+`	Concatenación	`s = s1 + s2`

Operadores

- Operadores de comparación
  - Se aplican sobre operandos del mismo tipo

Operador	Descripción	Ejemplo
==	Igualdad	x == y
!=	Desigualdad	x != y
>	Mayor que	x > y
<	Menor que	x < y
>=	Mayor o igual a	x >= y
<=	Menor o igual a	**x <= y**

Operadores

- Operadores lógicos
  - Se aplican sobre operandos boolean

Operador	Descripción	Ejemplo
`&&`	And	`x < 5 && x < 10`
`||`	Or	`x < 5 || x > 10`
`!`	Not	`!(x < 5 && x < 10)`

  - Long circuit ó short circuit ???

Operadores

- Operadores bitwise (bit a bit)
  - Se aplica sobre tipos numéricos

Operador	Descripción	Ejemplo
`&`	And	`x & y`
`|`	Or	`x | y`
`^`	Xor	`x ^ b`
`<<`	Shift a izquierda	`x << 2`
`>>`	Shift a derecha	**`x >> 2`**

Asignación

- Es una instrucción o construcción del lenguaje
  (no es un operador)
- Permite asignar valor a variables y constantes (en su declaración)
- Sintaxis:

  variable = expresión (del mismo tipo que la variable)

```
x = x + y * 10
```

Asignación

Instrucción	Ejemplo	Equivalente a
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	**x = x << 3**

Operadores (Asignación??)

- Operadores de incremento y decremento

Operador	Descripción	Ejemplo
++	Incremento	x++
--	Decremento	x--