Seminario de Lenguajes opción Go

Raúl Champredonde

Seminario de Lenguajes opción Go

- Estructuras de control
- Funciones
- Package “fmt”

Estructuras de control

- Secuencia

```
x := 5
fmt.Println(x)
x++
fmt.Println(x)
```

ó

```
x := 5; fmt.Println(x); x++; fmt.Println(x)
```

Estructuras de control

- Iteración

```
for {
   // secuencia
}
```

```
sum := 1
for sum < 1000 {
   sum += sum
}
```

```
sum := 0
for i := 0; i < 10; i++ {
   sum += i
}
```

```
sum := 1
for ; sum < 1000; {
   sum += sum
}
```

```
for i := 1; i < 500; {
   i += i
   fmt.Println(i)
}
```

Estructuras de control

- Iteración (repeat / do-while)

```
i := 0
for {
   i++
   if i >= 10 {
      break
   }
}
```

Estructuras de control

- Selección “if”

```
if x > y {
   fmt.Println(x)
   fmt.Println(y)
}
```

```
if x < y {
   fmt.Println(x)
} else {
   fmt.Println(y)
}
```

```
if x > y && x > z {
   fmt.Println("x")
} else if y > x && y > z {
   fmt.Println("y")
} else {
   fmt.Println("z")
}
```

Estructuras de control

- Selección “if” con sentencia de inicialización

```
if [variable := expresión;] condición {
   sentencias
} [else if condición {
   sentencias
}] [else {
   sentencias
}]
```

```
x := 3.0
n := 2.0
lim := 10.0

if v := math.Pow(x, n); v < lim {
   fmt.Println(v)
} else {
   fmt.Println(lim)
}
```

Estructuras de control

- Selección “switch”

```
switch [[variable := expresión;] selector] {
  {case expresión:
    sentencias}
  default:
    sentencias
}
```

```
package main

import (
  "fmt"
  "runtime"
)

func main() {
  switch runtime.GOOS {
    case "dar" + "win":
      fmt.Println("OS X.")
    case "li" + "nux":
      fmt.Println("Linux.")
    default:
      fmt.Println("Other")
  }
}
```

Estructuras de control

- Selección “switch” con sentencia de inicialización

```
switch [[variable := expresión;] selector] {
  {case expresión:
    sentencias}
  default:
    sentencias
}
```

```
switch os := runtime.GOOS; os {
  case "darwin":
    fmt.Println("OS X.")
  case "linux":
    fmt.Println("Linux.")
  default:
    fmt.Println("Other")
}
```

Estructuras de control

- Selección “switch” sin selector

```
switch [[variable := expresión;] selector] {
  {case expresión:
    sentencias}
  default:
    sentencias
}
```

```
package main

import (
  "fmt"
  "time"
)

func main() {
  t := time.Now()
  switch {
    case t.Hour() < 12:
      fmt.Println("Good morning!")
    case t.Hour() < 17:
      fmt.Println("Good afternoon.")
    default:
      fmt.Println("Good evening.")
  }
}
```

Funciones

`func nombreFuncion() {`
`  fmt.Println("Esta es una función")`
`}`
	`nombreFuncion()`

`func add(x int, y int) {`
`  fmt.Println(x + y)`
`}`
	`add(2, 3)`

`func add(x, y int) {`
`  fmt.Println(x + y)`
`}`
	`add(2, 3)`

Funciones

```
func add(x, y int) int {
  return x + y
}
```

```
z := add(2, 3)
```

```
func swap(x int, y int) (int, int) {
  return y, x
}
```

```
a, b = swap(a, b)
```

```
func swap(x1 int, y1 int) (x2, y2 int)
{
  x2, y2 = y1, x1
  return
}
```

```
a, b = swap(a, b)
```

Package “fmt”

`func Print(...) (n int, err error)`

Pone espacio entre argumentos excepto para strings	`const name, age = "Kim", 22`
	`fmt.Print(name, " is ", age, " years old.\n")`
	`// Kim is 22 years old.`

`func Println(...) (n int, err error)`

Pone espacio entre argumentos incluso para strings y agrega un “newline”	`const name, age = "Kim", 22`
	`fmt.Println(name, "is", age, "years old.")`
	`// Kim is 22 years old.`

`func Printf(format string, ...) (n int, err error)`

Usa “marcas” o “verbos”	`const name, age = "Kim", 22`
	`fmt.Printf("%s is %d years old.\n", name, age)`
	`// Kim is 22 years old.`

Package “fmt” (marcas/verbos)

```
i := 42
s := "Pepe“
b := true
```

General	%v	Formato predeterminado	fmt.Printf("%v %v", i, s)	42 Pepe
	%#v	Valor representado en sintaxis Go	fmt.Printf("%#v %#v", i, s)	42 "Pepe"
	%T	Tipo representado en sintaxis Go	fmt.Printf("%T %T", i, s)	int string
	%%	%	fmt.Printf("%v %v %%", s, i)	Pepe 42 %
	\n \r\n	new line	fmt.Printf("%v\n%v", i, s)	5 pepe
	\t	tab	fmt.Printf("%v\t%v", i, s)	5 pepe

Package “fmt” (marcas/verbos)

			i := 128578	
Integer	%d	Entero en decimal	fmt.Printf("%d", i)	128578
	%b	Entero en binario	fmt.Printf("%b", i)	11111011001000010
	%x	Entero en hexadecimal (min.)	fmt.Printf("%x", i)	1f642
	%X	Entero en hexadecimal (may.)	fmt.Printf("%X", i)	1F642
	%o	Entero en octal	fmt.Printf("%o", i)	373102
	%O	Entero en octal (con 0o)	fmt.Printf("%O", i)	0o373102
	%c	Carácter Unicode	fmt.Printf("%c", i)	🙂
	%q	Carácter Unicode con comillas simples	fmt.Printf("%q", i)	'🙂'
	%U	Formato Unicode	fmt.Printf("%U", i)	U+1F642

Package “fmt” (marcas/verbos)

			i := "Pepe"
String	%s	Valor normal	fmt.Printf("%s", i) Pepe
	%q	Valor encomillado	fmt.Printf("%q", i) "Pepe"
	%x	Base 16 minúscula	fmt.Printf("%x", i) 50657065
	%X	Base 16 mayúscula	fmt.Printf("%X", i) 50657065

Package “fmt” (marcas/verbos)

`pi := math.Pi`

Float	%e	Notación científica (min.)	`fmt.Printf("%e", pi)`	3.141593e+00
	%E	Notación científica (may.)	`fmt.Printf("%E", pi)`	3.141593E+00
	%f	Con decimales, sin exponente	`fmt.Printf("%f", pi)`	3.141593
	%F	Idem %f	`fmt.Printf("%F", pi)`	3.141593
	%g	%e para exponentes grandes o %f en caso contrario	`fmt.Printf("%g", pi)` `fmt.Printf("%g", pi * 1e+6)`	3.141592653589793 3.1415926535897930e+06
	%G	%E para exponentes grandes o %F en caso contrario	`fmt.Printf("%G", pi)` `fmt.Printf("%g", pi * 1e+6)`	3.141592653589793 3.1415926535897930e+06

Package “fmt” (marcas/verbos)

Equivalencias de `%v`	`bool`	`%t`
	`int, int8, etc.`	`%d`
	`uint, uint8, etc.`	`%d`
	`float32, complex32, etc.`	`%g`
	`string`	`%s`

Package “fmt” (width, precision)

```
i := 123
f := 123.12
```

`fmt.Printf("%d\n", i)`	`123`	`fmt.Printf("%+d\n", i)`	`+123`
`fmt.Printf("%6d\n", i)`	`   123`	`fmt.Printf("%+6d\n", i)`	`  +123`
`fmt.Printf("%06d\n", i)`	`000123`	`fmt.Printf("%+06d\n", i)`	`+00123`
`fmt.Printf("%f\n", f)`	`123.120000`	`fmt.Printf("%+f\n", f)`	`+123.120000`
`fmt.Printf("%0f\n", f)`	`123.120000`	`fmt.Printf("%+0f\n", f)`	`+123.120000`
`fmt.Printf("%8.2f\n", f)`	`  123.12`	`fmt.Printf("%+8.2f\n", f)`	`+123.12`
`fmt.Printf("%08.2f\n", f)`	`00123.12`	`fmt.Printf("%+08.2f\n", f)`	`+0123.12`

Package “fmt” (flags)

- Investigar:
  - "-" (ej.: `%-d`, `%-6d`, `%-06d`, `%-f`, `%-8.2f`, `%-08.2f`)
  - "#"
  - " "
  - "%.2f"
  - "%9.f"

Package “fmt”

```
const name, age = "Kim", 22
```

```
s := fmt.Sprintf("%s is %d years old.\n", name, age)
```

```
s := fmt.Sprint(name, " is ", age, " years old.\n")
```

```
s := fmt.Sprintln(name, "is", age, "years old.")
```

```
Kim is 22 years old.
```

Package “fmt”

```
func Scan(...) (n int, err error)
```

```
var mensaje string
n, e := fmt.Scan(&mensaje)
```

```
func Scanf(format string, ...) (n int, err error)
```

```
var (nom string; ape string; tel int)
n, e := fmt.Scanf("%s %s %d", &nom, &ape, &tel)
if e != nil {
    fmt.Printf("Error: %s", e)
} else {
    fmt.Printf("Todo bien, recibimos %d argumentos: %s, %s, %d", n, nom, ape, tel)
}
```

```
func Scanln(...) (n int, err error)
```

```
var nom, ape string
n, e := fmt.Scanln(&nom, &ape)
```

Package “fmt”

```
func Sscan(str string, ...) (n int, err error)
```

```
func Sscanln(str string, ...) (n int, err error)
```

```
func Sscanf(str string, format string, ...) (n int, err error)
```

```
var x, y string

n, e := fmt.Sscan("100\n200", &x, &y)
n, e = fmt.Sscan("300 400", &x, &y)
n, e = fmt.Sscanf("500 600", "%s %s", &x, &y)
n, e = fmt.Sscanln("700\n800", &x, &y)
n, e = fmt.Sscanln("900 1000\n", &x, &y)
n, e = fmt.Sscanln("1100 1200", &x, &y)
```

```
n: 2 e: <nil> Sscan x: 100 y: 200
n: 2 e: <nil> Sscan x: 300 y: 400
n: 2 e: <nil> Sscanf x: 500 y: 600
n: 1 e: unexpected newline Sscanln x: 700 y: 600
n: 2 e: <nil> Sscanln x: 900 y: 1000
n: 2 e: <nil> Sscanln x: 1100 y: 1200
```