

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

<pre>for { // secuencia }</pre>	<pre>sum := 1 for sum < 1000 { sum += sum }</pre>	<pre>sum := 0 for i := 0; i < 10; i++ { sum += i }</pre>
		<pre>sum := 1 for ; sum < 1000; { sum += sum }</pre>

Estructuras de control

- Iteración (for)

```
for i, j := 0, 10; i <= j; i, j = i+1, j-1 {  
    fmt.Println(n, ":", i, "-", j)  
}
```

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)  
}  
  
fmt.Println(v)
```


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)
```

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)
```

Package “fmt”

```
package main

import "fmt"

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

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.
```

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.
```

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	<code>%v</code>	Formato predeterminado	<code>fmt.Printf("%v %v", i, s)</code>	42 Pepe
	<code>%#v</code>	Valor representado en sintaxis Go	<code>fmt.Printf("%#v %#v", i, s)</code>	42 "Pepe"
	<code>%T</code>	Tipo representado en sintaxis Go	<code>fmt.Printf("%T %T", i, s)</code>	int string
	<code>%%</code>	%	<code>fmt.Printf("%v %v %%", s, i)</code>	Pepe 42 %
	<code>\n</code> <code>\r\n</code>	new line	<code>fmt.Printf("%v\n%v", i, s)</code>	5 pepe
	<code>\t</code>	tab	<code>fmt.Printf("%v\t%v", i, s)</code>	5 pepe
Boolean	<code>%t</code>	"true" o "false"	<code>fmt.Printf("%t", b)</code>	true

Package “fmt” (marcas/verbos)

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

Package “fmt” (marcas/verbos)

		<code>i := "Pepe"</code>		
String	<code>%s</code>	Valor normal	<code>fmt.Printf("%s", i)</code>	Pepe
	<code>%q</code>	Valor encomillado	<code>fmt.Printf("%q", i)</code>	"Pepe"
	<code>%x</code>	Base 16 minúscula	<code>fmt.Printf("%x", i)</code>	50657065
	<code>%X</code>	Base 16 mayúscula	<code>fmt.Printf("%X", i)</code>	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)	3.141592653589793
			fmt.Printf("%g", pi * 1e+6)	3.141592653589793e+06
	%G	%E para exponentes grandes o %F en caso contrario	fmt.Printf("%G", pi)	3.141592653589793
			fmt.Printf("%g", pi * 1e+6)	3.141592653589793e+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
```

<code>fmt.Printf("%d\n", i)</code>	123 ---	<code>fmt.Printf("%+d\n", i)</code>	+123 ----
<code>fmt.Printf("%6d\n", i)</code>	123 -----	<code>fmt.Printf("%+6d\n", i)</code>	+123 -----
<code>fmt.Printf("%06d\n", i)</code>	000123 -----	<code>fmt.Printf("%+06d\n", i)</code>	+00123 -----
<code>fmt.Printf("%f\n", f)</code>	123.120000 -----	<code>fmt.Printf("%+f\n", f)</code>	+123.120000 -----
<code>fmt.Printf("%0f\n", f)</code>	123.120000 -----	<code>fmt.Printf("%+0f\n", f)</code>	+123.120000 -----
<code>fmt.Printf("%8.2f\n", f)</code>	123.12 -----	<code>fmt.Printf("%+8.2f\n", f)</code>	+123.12 -----
<code>fmt.Printf("%08.2f\n", f)</code>	00123.12 -----	<code>fmt.Printf("%+08.2f\n", f)</code>	+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.Sprintln(name, "is", age, "years old.")
```

Package “fmt”

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

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

```
    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)  
    }  
}
```

```
    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)
```

```
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
```