INTRODUCCIÓN A LA CLASE

Go Bases

Objetivos de esta clase

Los objetivos de la clase son:

- Comprender y Utilizar el paquete fmt
- Comprender y Utilizar el paquete os
- Comprender y Utilizar el paquete io

¡Vamos por ello!

PACKAGE FMT

Go Bases

// ¿Qué es el package fmt?

El package “fmt” nos permite imprimir por pantalla valores y formatearlos, tanto para imprimirlo por pantalla como para trabajarlo como texto.

Caracteres de escape

Los caracteres de escape son utilizados en Go al momento de imprimir un string para “escapar del string”.

Go no imprimirá en pantalla cuando encuentre un carácter de escape, lo que hará es formatear según el carácter de escape utilizado.

Por ejemplo si en una instrucción de impresión encuentra la directiva
“\n” Go interpreta que debe imprimir un salto de línea en ese lugar.

Caracteres de escape más utilizados

CARÁCTER DE ESCAPE	INTERPRETACIÓN
\n	salto de línea
\\	barra invertida
\t	tab horizontal
\v	tab vertical

Verbos de impresión

Algunas funciones en el package fmt aceptan verbos de impresión. Estos verbos de impresión le indican a Go de qué manera imprimir una variable.

Cuando encuentra un verbo de impresión, Go lo reemplaza por la variable correspondiente de manera que el primer verbo de impresión corresponde a la primer variable, y así sucesivamente.

Estos verbos de impresión le indican a Go como tratar a esa variable. Por ejemplo para imprimir por pantalla el tipo de una variable usamos el verbo de impresión %T.

Verbos de impresión más utilizados

%v	valor en formato estándar
%T	tipo de dato del valor a imprimir
%t	bool
%s	string
%f	punto flotante
%d	entero decimal
%**b**	un entero binario
%o	octal
%c	imprime caracteres
%p	dirección de memoria

Print

// Func

IT BOARDING

BOOTCAMP

.Print( )

La función Print(a ...interface{ }) (n int, err error) toma como parámetro n cantidad de valores de cualquier tipo (interface { }) y escribe al standard output (normalmente la pantalla).
La función devuelve el número de bytes escritos, y un error en caso de encontrar uno.

Es convención ignorar los valores retornados por la función Print( ).

.Print( )

ULO

En este ejemplo definimos e inicializamos las constantes nombre y edad. Ya que la función Print( ) recibe n número de parámetros y de cualquier tipo, podemos pasarle estas constantes, y concatenarlas con cadenas definidas dentro de la función.

“\n” es un carácter de escape que indica un salto de línea.

```
{}
const nombre, edad = "Kim", 22
fmt.Print(nombre, " tiene ", edad, " años de edad.\n")
```

.Println( )

La función Println( ) funciona exactamente igual que print( ). La única diferencia es que siempre agrega un salto de línea al final de la impresión.

```
{}
const nombre, edad = "Kim", 22
fmt.Println(nombre, " tiene ", edad, " años de edad.")
```

.Printf( )

La función Printf(format string, a ...interface{ }) (n int, err error) toma como primer parámetro un string en donde se colocan los verbos de impresión y como segundo parámetro n cantidad de valores de cualquier tipo (interface {}) y escribe al standard output (normalmente la pantalla).
La función devuelve el número de bytes escritos, y un error en caso de encontrar uno.

Es convención ignorar los valores retornados por la función printf( ).

.Printf( )

En este ejemplo modificamos el ejemplo anterior usando printf( ) en vez de print( ).
Como primer parámetro pasamos el string de formateo que incluirá los verbos de impresión que al momento de imprimirse serán reemplazados por las variables de los parámetros subsiguientes.
Pasamos **%s** porque nombre es un string y **%d** porque edad es un entero.

```
{}
const nombre, edad = "Kim", 22
fmt.Printf("%s tiene %d años de edad.\n", nombre, edad)
```

.Printf( )

También podemos definir que la impresión tenga una cantidad X de caracteres, en caso del valor a imprimir sea menor a esa cantidad. De ser así rellenara los valores faltantes con espacios en blanco.
Si el valor a imprimir supera la cantidad que indicamos no le dará importancia e imprimirá el valor normalmente.

```
{}
fmt.Printf("%10d", 12222)
fmt.Printf("%10s", "aa")
```

.Printf( )

Cuando deseamos imprimir un flotante podemos indicarle la cantidad de valores que queramos imprimir después de la coma.
Lo hacemos indicandolo con un punto y la cantidad.
En el siguiente ejemplo veremos cómo imprimimos solo los primeros 2 valores por pantalla, quedando 12222.12

```
{}
fmt.Printf("%10.2f", 12222.123123)
fmt.Printf("%.2f", 12222.123123)
```

Sprint

//Func

IT BOARDING

BOOTCAMP

.Sprint( )

La función Sprint(a ...interface{ }) string toma como parámetro n variables de cualquier tipo y las formatea como un string de manera estándar según su tipo.
El valor de retorno es la concatenación de estas variables condensadas en un string.

En este ejemplo usamos la función Sprint( ) para guardar en una variable la concatenación de todas las variables pasadas por parámetro y luego usamos la función Print( ) para imprimir la variable generada.

```
{}
const nombre, edad = "Kim", 22
res := fmt.Sprint(nombre, " tiene ", edad, " años de edad.\n")
fmt.Print(res)
```

.Println( )

La función Println( ) funciona exactamente igual que la función Print( ) con la diferencia de que agrega un salto de línea automáticamente al final del string resultante.

```
{}
const nombre, edad = "Kim", 22
res := fmt.Println(nombre, " tiene ", edad, " años de edad.")
fmt.Print(res)
```

.Sprintf( )

La función Sprintf(format string, a ...interface{ }) string toma como parámetro un string de formateo al igual que la función Printf( ) y n variables de cualquier tipo y las formatea según el verbo de impresión indicado en el string de formateo.
El valor de retorno es la concatenación de estas variables condensadas en un string.

.Sprintf( )

En este ejemplo modificamos el anterior usandao Sprintf( ) en vez de Sprint( ).
La diferencia es que le pasamos un string con los verbos de impresión en donde irían las variables de los parámetros subsiguientes. El verbo de impresión indicaría a Go de qué manera tratar a esas variables a la hora de formatearlas.

```
{}
const nombre, edad = "Kim", 22
res := fmt.Sprintf("%s tiene %d años de edad.\n", nombre, edad)
fmt.Print(res)
```

Al pasar %s le decimos que trate a la primer variable como string y con %d que trate a la siguiente como int.

// Para concluir

Hemos visto las funciones principales del paquete fmt. Hay otras herramientas que no vimos en esta presentación. Te invitamos a ver la documentación oficial en caso de querer profundizar más.

PACKAGE OS

Go Bases

// ¿Qué veremos en esta sección?

Introduciremos las funciones principales y más usadas del paquete os.

// ¿Qué es el package os?

El package “os” nos permite ejecutar y utilizar funcionalidades del Sistema Operativo.

Funcionalidades Sistema Operativo

Podemos utilizar funcionalidades del sistema operativo como por ejemplo:

- Conocer la información respectiva de un determinado archivo
  - Nombre
  - Tamaño
  - Fecha
- Variables de entorno
  - Leer
  - Escribir
  - Eliminar
- Leer un archivo

Stat

// Func

IT BOARDING

BOOTCAMP

.Stat( )

La función Stat(name string) recibe por parámetro la ubicación de un archivo o directorio en formato string y devuelve un objeto FileInfo

```
filename := "./miArchivo"
f, err := os.Stat(filename)
```

.Stat( )

El objeto FileInfo contiene información con respecto al nombre, tamaño, si es archivo o directorio, fecha de modificación y permisos

```
{}
fmt.Println("Es un directorio: ", f.IsDir())
fmt.Println("Nombre del archivo/directorio: ", f.Name())
fmt.Println("Tamaño del archivo en Bytes: ", f.Size())
fmt.Println("Fecha y Hora del archivo: ", f.ModTime())
fmt.Println("Permisos del archivo", f.Mode())
```

Consideraciones

En caso de no encontrar el archivo nos devolverá un error y el objeto FileInfo estará en nil.

Getenv

//Func

IT BOARDING

BOOTCAMP

.Getenv( )

La función Getenv(key string) nos permitirá acceder a las variables de entorno del sistema, le pasaremos por parámetro la variable a la que deseamos acceder y nos devolverá su valor:

```
valor := os.Getenv("HOME")
```

Consideraciones

En caso de tratar de acceder a una variable de entorno que no existe nos devolverá una cadena de texto vacía, si necesitamos saber si una variable existe y tiene el valor en blanco o no existe podemos utilizar LookupEnv.

LookupEnv

//Func

IT BOARDING

BOOTCAMP

.LookupEnv( )

La función LookupEnv(key string) es equivalente a la función Getenv( ) con la diferencia que retorna 2 valores:

- El valor de la variable de entorno.
- Un booleano para determinar si la variable existe o no.

```
valor, existe := os.LookupEnv("HOME")
```

Consideraciones

En caso que la variable exista y esté vacía nos devolverá el valor vacio y el booleano en verdadero.
En caso que la variable no exista nos devolverá el valor en vacío y el booleano en falso

Setenv

// Func

IT BOARDING

BOOTCAMP

.Setenv( )

La función Setenv(key, value string) modifica el valor de una variable de entorno recibiendo el nombre y el valor a asignar. Retornará un error en caso de algún inconveniente.

```
{}  err := os.Setenv("MI_VARIABLE", "mi valor")
```

Consideraciones

En caso que la variable exista y esté vacía nos devolverá el valor vacio y el booleano en verdadero.
En caso que la variable no exista nos devolverá el valor en vacío y el booleano en falso.

Unsetenv

//Func

IT BOARDING

BOOTCAMP

.Unsetenv( )

La función Unsetenv(key string) recibe el nombre de la variable de entorno y la elimina, retorna error en caso de algún inconveniente.

```
{}  err := os.Unsetenv("MI_VARIABLE")
```

// Para concluir

Hemos visto las funciones principales del paquete os. Hay otras herramientas que no incluimos en esta presentación. Te invitamos a ver la documentación oficial en caso de querer profundizar más.

PACKAGE IO

Go Bases

// ¿Qué es el package io?

El package “io” nos permite utilizar las funcionalidades primitivas de Entrada y Salida, como Leer y Escribir archivos, entre otras.

// Cuidado!

Debido a que las funcionalidades del paquete envuelven operaciones de bajo nivel, no nos dará seguridad si utilizamos paralelismo. Como por ejemplo, utilizar varios hilos para manipular un archivo.

ioutil

// Package

IT BOARDING

BOOTCAMP

ioutil

A partir de la versión 1.16 de Go, io nos proporciona el paquete ioutil. Este paquete nos ayuda a utilizar las funcionalidades más utilizadas de io de una manera más simple.

El paquete ioutil está dentro del paquete io, para importarlo lo hacemos de la siguiente manera:

```
{}
import (
    "io/ioutil"
)
```

ReadFile

// Func

IT BOARDING

BOOTCAMP

.ReadFile( )

La función ReadFile(filename string) recibe como parámetro la dirección y nombre del archivo en formato texto y nos devuelve el contenido del archivo en bytes o un error en caso que lo haya .

```
{}  dat, err := ioutil.ReadFile("./miArchivo.txt")
```

WriteFile

// Func

IT BOARDING

BOOTCAMP

.WriteFile( )

La función WriteFile(filename string, data []byte, perm fs.FileMode) recibe como parámetro la dirección y nombre del archivo en formato texto, su contenido en formato bytes y el permiso que queramos asignarle.
No devuelve un error en caso que lo haya.

```
{}
d1 := []byte("hello\ngo\n")
err := ioutil.WriteFile("./dat1", d1, 0644)
```

// Para concluir

Hemos visto las funciones principales del paquete io. Hay otras herramientas que no vimos en esta presentación. Te invitamos a ver la documentación oficial en caso de querer profundizar más.

Gracias.

IT BOARDING

BOOTCAMP