Cheat Sheet Go

En esta lectura te llevarás algunos tópicos que siempre vale la pena tener a la mano al momento de programar en Go, además te dejo algunos tips extras.

- Hola mundo

```
package main

import "fmt"

func main(){
    fmt.Println("Hola mundo")
}
```

- ¿Hacer una impresión en consola rápida?

```
package main

func main(){
    print("Hola")
}
```

- Importar una librería sin usarla

```
package main

/*
    Hazlo solo y únicamente cuando la librería externa
    que estés usando lo pida explícitamente
*/
import (
    "fmt"
    _ "math"
)

func main(){
    fmt.Println("Hola mundo")
}
```

- Agregar un alias a un import (no suele usarse, pero es bueno saberlo)

```
package main

import (
  "fmt"
  mth "math"
)

func main() {
  fmt.Println(mth.Pi)
}
```

- Diferentes formas de declarar variables

```
v := 12
var v int = 12
var v int
```

- Zero values de primitivos

```
var a int // 0
var b float64 // 0
var c string // ""
var d bool // false
```

- Incremental y decremental

```
x++ // Suma 1 a x
x-- // Resta 1 a x
```

- Imprimir tipo de variables (hay otras formas, pero esta es la más fácil)

```
a := 2
fmt.Printf("%T", a)
```

- Función para tomar los errores (ahorra mucho código)

```
func isError(e error) {
    if e != nil {
        log.Fatal(e)
    }
}

// Ejemplo de uso
func main() {
  _, err := strconv.Atoi("53a")
  isError(err)
}
```

- Arrays vs Slices

```
// Array
var myList [2]int

// Slice
var myList2 []int
```

- Slice de interfaces (Úsalo con sabiduría)

```
// Permite guardar diferentes tipos de datos en un mismo slice
myList := []interface{}{"Hola", 12, 4.90}

// Iterar sobre los distintos tipos de datos de ese slice
for _, v := range myList {
```

```
    switch v.(type) {
    case int:
        fmt.Println("Es int")
    case string:
        fmt.Println("Es string")
    case float64:
        fmt.Println("Es float64")
    }
}
```

- Asegurarnos si un key existe en el map

```
m := make(map[string]int)

m["hola"] = 1

// Nota, usalmente se usa "ok" para recibir la segunda variable
value, ok := m["hello"]

/*
Si existe, ok será "true"
Si no existe, ok será "false"

En este caso, ok es "false" porque no existe.
*/
```

- Punteros

```
a := 10 // Variable int
b := &a // "b" es el puntero de "a"
c := *b // "c" adquiere el valor del puntero de "b", es decir toma el mismo
valor de "a"
```

- Comandos de Go modules

```
// Inicializar un proyecto
go mod init path_del_proyecto

// Verificar que el código externo no esté corrupto
go mod verify

// Reemplazar fuente del código
go mod edit -replace path_del_repo_online=path_del_repo_en_local

// Quitar el replace
go mod edit -dropreplace path_del_repo_online

// Empaquetar todo el código de terceros que usa nuestro código
go mod vendor

// Eliminar todos los paquetes externos que no estemos usando
go mod tidy

// Aprender más de go modules
go help mod
```

- Nota personal

Aún tienes un largo camino por recorrer. Pero lo que más quiero que te lleves de este curso son tres cosas: Practica, estudia y participa en la comunidad de Go.