

Go: Taller práctico 1

Profesor Yerson Ferney Porras García

Nota: Los fragmentos de código que se mostrarán serán definidos dentro de la función main, es decir que estarán ubicados dentro como se muestra en la siguiente imagen a menos que se indique algo particular:

```
1  package main
2
3  import (
4      "fmt"
5  )
6
7  func main () {
8
9      //El código irá aquí...
10
11 }
```

Tema Verificación de tipos de variables:

Podemos saber el tipo de una variable de la siguiente manera: importando el paquete "reflect" el cual es un paquete por defecto de Go:

```
import (
    "fmt"
    "reflect"
)
```

```
var nombre string = "Juan Pepe"
var apellido string
nick := "Pepe grillo"
var edad int = 20
var altura float32
peso := 70.0
inscrito := true

fmt.Println(reflect.TypeOf(nombre))
fmt.Println(reflect.TypeOf(apellido))
fmt.Println(reflect.TypeOf(nick))
fmt.Println(reflect.TypeOf(edad))
fmt.Println(reflect.TypeOf(altura))
fmt.Println(reflect.TypeOf(peso))
fmt.Println(reflect.TypeOf(inscrito))
```

```
string
string
string
int
float32
float64
bool
```

Tema Arreglos:

En Go Podemos declarar arreglos o también conocidos como arrays. Para esto definimos el tamaño del arreglo junto con el tipo lo cual indicará el tipo de datos homogéneos que permitirá almacenar. El tamaño de los arreglos no se puede modificar.

```
var alumnos [4]string
```

Usando la palabra reservada “var” definiremos la variable “alumnos” como arreglo unidimensional de tamaño 4 y de tipo “string”.

Para insertar valores al arreglo lo haremos por medio de su índice que deberá estar dentro del tamaño definido:

```
alumnos[0] = "Juan"  
alumnos[1] = "Pepe"  
alumnos[2] = "Maria"  
alumnos[3] = "Enrique"
```

Si intentamos acceder a una posición ya sea para obtener o asignar un dato obtendremos un error.

```
alumnos[4] = "Andrés"
```

```
# command-line-arguments  
.\Main.go:16:10: invalid argument: index 4 out of bounds [0:4]
```

Podemos declarar e inicializar arreglos directamente:

```
var arr1 = [3]int{1,2,3}  
arr2 := [5]int{4,5,6,7,8}  
  
fmt.Println(arr1)  
fmt.Println(arr2)
```

```
[1 2 3]  
[4 5 6 7 8]
```

También con la declaración del tamaño implícito

```
var arr1 = [...]int{1,2,3}  
arr2 := [...]int{4,5,6,7,8}  
  
fmt.Println(arr1)  
fmt.Println(arr2)
```

```
[1 2 3]  
[4 5 6 7 8]
```

Adicionalmente, podemos declarar un arreglo e inicializarlo parcialmente

```
arr1 := [5]int{} //no inicializado
arr2 := [5]int{1,2} //parcialmente inicializado
arr3 := [5]int{1,2,3,4,5} //completamente inicializado

fmt.Println(arr1)
fmt.Println(arr2)
fmt.Println(arr3)
```

```
[0 0 0 0 0]
[1 2 0 0 0]
[1 2 3 4 5]
```

También podemos inicializar parcialmente en ubicaciones específicas:

```
arr1 := [5]int{1:10,2:40}
fmt.Println(arr1)
```

```
[0 10 40 0 0]
```

Aquí se inicializan los elementos en los índices 1 y 2 con los valores 10 y 40, respectivamente.

Por último, podemos obtener el tamaño del arreglo con la función len.

```
arr1 := [4]string{"Volvo", "BMW", "Ford", "Mazda"}
arr2 := [...]int{1,2,3,4,5,6}

fmt.Println(len(arr1))
fmt.Println(len(arr2))
```

```
4
6
```

Subarreglos:

Podemos acceder a un subconjunto de elementos del arreglo usando “i:e” donde “i” representa la posición de inicio del arreglo y “e” la posición final no incluyente.

```
fmt.Println(arr1[0:4])
fmt.Println(arr1[:3])
fmt.Println(reflect.TypeOf(arr1[0:4]))
fmt.Println(reflect.TypeOf(arr1[:3]))
```

```
[Volvo BMW Ford Mazda]
[Volvo BMW Ford]
[]string
[]string
```

Si no se especifica un valor “i” se entiende que va desde el inicio, es decir i=0. Ahora, si no se especifica el valor “e” va hasta el último, es decir, e=len(arr1).

Nota: Estos subarreglos son retornados como segmentos, este tema se verá más adelante en el documento.

```
fmt.Println(reflect.TypeOf(arr1) == reflect.TypeOf(arr1))
fmt.Println(reflect.TypeOf(arr1) == reflect.TypeOf(arr1[:]))
```

```
true
false
```

Tema: Conversiones de tipo

En algunos casos podremos cambiar el tipo (cast) de un dato a otro de manera directa

```
entero := 23
fmt.Println(reflect.TypeOf(float64(entero)))

flotante := 23.3
fmt.Println(reflect.TypeOf(int8(flotante)))
fmt.Println(int8(flotante))
```

```
float64
int8
23
```

Sin embargo, habrá casos donde no se podrá de manera directa:

```
cadena := "23"
fmt.Println(reflect.TypeOf(float64(cadena)))
fmt.Println(reflect.TypeOf(int8(cadena)))
```

```
# command-line-arguments
.\Main.go:20:37: cannot convert cadena (variable of type string) to type float64
.\Main.go:21:34: cannot convert cadena (variable of type string) to type int8
```

Por lo cual, se requerirá de módulos adicionales como "strconv", el cual es un módulo que nos permite convertir strings a algún otro tipo:

```
cadena := "23"
intVar, err := strconv.Atoi(cadena)
fmt.Println(intVar, err, reflect.TypeOf(intVar))
```

En este caso la función retorna el int y el error. Dado el caso no se pueda hacer la conversión, el error es útil para crear un manejador de error:

```
cadena2 := "23s"
intVar2, err := strconv.Atoi(cadena2)
if err != nil {
    fmt.Println("No se puede convertir")
}
fmt.Println(cadena2, intVar2)
```

```
No se puede convertir
23s 0
```

Nota: La función Atoi proviene de una función base llamada ParseInt la cual recibe como parámetros la cadena, base del entero y tamaño en memoria (bits). Se suele usar la función Atoi por facilidad y rapidez en la codificación.

func ParseInt(s string, base int, bitSize int) (i int64, err error)

```
strVar := "100"

intVar, err := strconv.ParseInt(strVar, 0, 8)
fmt.Println(intVar, err, reflect.TypeOf(intVar))

intVar, err = strconv.ParseInt(strVar, 0, 16)
fmt.Println(intVar, err, reflect.TypeOf(intVar))

intVar, err = strconv.ParseInt(strVar, 0, 32)
fmt.Println(intVar, err, reflect.TypeOf(intVar))

intVar, err = strconv.ParseInt(strVar, 0, 64)
fmt.Println(intVar, err, reflect.TypeOf(intVar))
```

```
100 <nil> int64
100 <nil> int64
100 <nil> int64
100 <nil> int64
```

un número en binario (base 2):

```
strVar := "11"
intVar, err := strconv.ParseInt(strVar, 2, 8)
fmt.Println(intVar, err, reflect.TypeOf(intVar))
```

```
3 <nil> int64
```

Otros ejemplos de conversiones:

```
esHombre := "true"
boolVal, _ := strconv.ParseBool(esHombre)
fmt.Println(boolVal, reflect.TypeOf(boolVal))
```

```
true bool
```

```
esHombre := true
strVal := strconv.FormatBool(esHombre)
fmt.Println(strVal, reflect.TypeOf(strVal))
```

```
true string
```

Tema: Funciones

Función sin retorno

```
func saludo(nombre string){
    fmt.Printf("Hola, %v. Bienvenido!", nombre)
}
```

Función con retorno

```
func saludo(nombre string) string{  
    mensaje := fmt.Sprintf("Hola, %v. Bienvenido!", nombre)  
    return mensaje  
}
```

Llamado a la función sin retorno

```
saludo(alumnos[0])
```

Llamado a la función con retorno

```
fmt.Println(saludo(alumnos[0]))
```

Funciones con múltiples retornos:

```
const Pi = 3.1416
```

```
func circulo(radio float64) (area float64, perimetro float64) {  
    area = Pi * radio * radio  
    perimetro = 2 * Pi * radio  
    return area, perimetro  
}
```

```
a, p := circulo(8)  
fmt.Println("El área del círculo es: ", a)  
fmt.Println("El perímetro del círculo es: ", p)
```

```
El área del círculo es: 201.0624  
El perímetro del círculo es: 50.2656
```

Tema Arreglos multidimensionales:

Así como existen los arreglos multidimensionales en otros lenguajes, también los tenemos en Go y su declaración no varía mucho de la declaración de unidimensionales, donde agregamos un par de “[]” por cada dimensión que queramos incluyendo dentro el tamaño de dicha dimensión. En la siguiente imagen se crea un arreglo bidimensional (alias matriz) de 2*2 (2 filas y 2 columnas). Luego se crea otro arreglo bidimensional de 2*3 inicializado con datos directamente.

```
var tabla0 [2][2] int  
var tabla1 = [2][3]int {{1,2,3},{3,4,3}}  
fmt.Println(tabla0)  
fmt.Println(tabla1)  
fmt.Println(tabla1[0][1])
```

```
[[0 0] [0 0]]  
[[1 2 3] [3 4 3]]  
2
```

Tema Segmentos (Slices):

Otra forma de manipular conjuntos de datos son los segmentos, los cuales son similares a los arreglos con la particularidad que no tienen un tamaño estático, no se define un tamaño en la declaración y podemos incrementar o decrementar la cantidad de elementos. Se definen con “[]” sin un valor de tamaño interno.

```
arr1 := [4]string{"Volvo", "BMW", "Ford", "Mazda"}
arr2 := [...]int{1,2,3,4,5,6}

fmt.Println(reflect.TypeOf(arr1))
fmt.Println(reflect.TypeOf(arr2))

slice1 := []string{"Volvo", "BMW", "Ford", "Mazda"}
slice2 := []int{1,2,3,4,5,6}
fmt.Println(reflect.TypeOf(slice1))
fmt.Println(reflect.TypeOf(slice2))
```

```
[4]string
[6]int
[]string
[]int
```

Los arreglos y los segmentos son de tipos diferentes

```
fmt.Println(reflect.TypeOf(arr1) == reflect.TypeOf(arr1))
fmt.Println(reflect.TypeOf(arr1) == reflect.TypeOf(arr2))
fmt.Println(reflect.TypeOf(slice1) == reflect.TypeOf(slice1))
fmt.Println(reflect.TypeOf(slice1) == reflect.TypeOf(slice2))
fmt.Println(reflect.TypeOf(arr1) == reflect.TypeOf(slice1))
```

```
true
false
true
false
false
```

Nota: se aborda como introducción de tema pero se abordará con más profundidad en la próxima clase.

Tema Condicional IF:

El condicional es muy similar el If de Java. En Go sigue la siguiente forma:

```
11     var edad uint8
12     fmt.Println("¿Cuál es tu edad?:")
13     fmt.Scanln(&edad)
14
15     if (edad >= 18) {
16         fmt.Println("Eres mayor de edad")
17     }
18
19 }
```

```
¿Cuál es tu edad?:
20
Eres mayor de edad
```

En la línea 13 se pide un dato por teclado, para lo cual se usa un apuntador a la variable con el carácter "&". Luego en la línea 15 realizamos el condicional if. Los paréntesis son opcionales.

Algo característico de Go es que podemos realizar dentro del if una asignación previa a la comparación:

Para el siguiente caso importamos el módulo "math".

```
func pow(x, n, lim float64) float64 {  
    if v := math.Pow(x, n); v < lim {  
        return v  
    }  
    return lim  
}
```

```
fmt.Println(  
    pow(3, 2, 10),  
    pow(3, 3, 20),  
)
```

```
9 20
```

En el anterior código, definimos una función llamada "pow" que recibe 3 parámetros de los cuales dos tienen el tipado implícito ("x" y "n") y un parámetro tiene el tipado explícito ("lim"). Allí dentro de la instrucción if declaramos e inicializamos una variable "v" que tendrá un alcance local para el if y con la cual podremos hacer la comparación ("v < lim").

También tenemos el "else". Muy similar a java.

```
if 7%2 == 0 {  
    fmt.Println("7 is even")  
} else {  
    fmt.Println("7 is odd")  
}  
  
if 8%4 == 0 {  
    fmt.Println("8 is divisible by 4")  
}  
  
if num := 9; num < 0 {  
    fmt.Println(num, "is negative")  
} else if num < 10 {  
    fmt.Println(num, "has 1 digit")  
} else {  
    fmt.Println(num, "has multiple digits")  
}
```

```
7 is odd  
8 is divisible by 4  
9 has 1 digit
```

Tema Apuntadores:

Los apuntadores son puntos de acceso a una ubicación específica de la memoria donde se alojará algún dato. Su uso es muy similar a C.


```

i, j := 42, 2701

p := &i          // apuntador a i
fmt.Println(*p) // lee i a través del apuntador p
*p = 21          // cambia el valor de i a través del apuntador p
fmt.Println(i)   // accede al nuevo valor de i
fmt.Println(p)   // accede al apuntador como tal

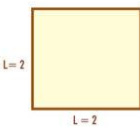
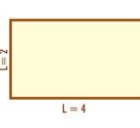
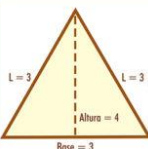
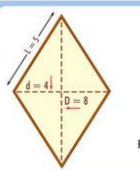
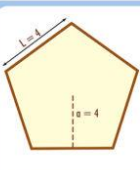
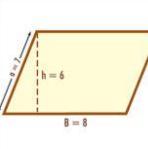
p = &j          // apuntador a j
*p = *p / 37     // divide j a través del apuntador
fmt.Println(j)

```

Ejercicios:

1. Crear una función que reciba una cadena de texto y retorne un true si es palíndromo y un false caso contrario.
2. Crear una función por cada área y perímetro de las siguientes figuras geométricas donde reciba como parámetro la(s) variable(s) requerida(s):

ÁREAS Y PERÍMETROS

ÁREA DEL CUADRADO  $\text{Área} = L \times L$ $\text{Perímetro} = L + L + L + L$ $\text{Área} = 2 \times 2 = 4 \text{ cm}^2$ $\text{Perímetro} = 4 + 4 + 4 + 4 = 16 \text{ cm}$	ÁREA DEL RECTÁNGULO  $\text{Área} = L \times l$ $\text{Perímetro} = L + L + l + l$ $\text{Área} = 2 \times 4 = 8 \text{ cm}^2$ $\text{Perímetro} = 2 + 2 + 4 + 4 = 12 \text{ cm}$	ÁREA DEL TRIÁNGULO  $\text{Área} = \frac{\text{Base} \times \text{Altura}}{2}$ $\text{Perímetro} = L + L + L$ $\text{Área} = \frac{3 \times 4}{2} = 6 \text{ cm}^2$ $\text{Perímetro} = 3 + 3 + 3 = 9 \text{ cm}$
ÁREA DEL ROMBO  $\text{Área} = \frac{D \times d}{2}$ $\text{Perímetro} = L + L + L + L$ $\text{Área} = \frac{8 \times 4}{2} = 16 \text{ cm}^2$ $\text{Perímetro} = 5 + 5 + 5 + 5 = 20 \text{ cm}$	ÁREA DEL PENTÁGONO  $\text{Área} = \frac{\text{Perímetro} \times \text{apotema}}{2}$ $\text{Perímetro} = 5 \times L$ $\text{Área} = \frac{20 \times 4}{2} = 40 \text{ cm}^2$ $\text{Perímetro} = 5 \times 4 = 20 \text{ cm}$	ÁREA DEL HEXÁGONO  $\text{Área} = \frac{\text{Perímetro} \times \text{apotema}}{2}$ $\text{Perímetro} = 6 \times L$ $\text{Área} = \frac{36 \times 5}{2} = 90 \text{ cm}^2$ $\text{Perímetro} = 6 \times 6 = 36 \text{ cm}$
ÁREA DEL CÍRCULO  $\text{Área} = \pi \times r^2$ $\text{Perímetro} = 2 \times \pi \times r$ $\text{Área} = 3,14 \times 3^2$ $= 3,14 \times 9 = 28,26 \text{ cm}^2$ $\text{Perímetro} = 2 \times \pi \times 3$ $= 2 \times 3,14 \times 3 = 18,8 \text{ cm}$	ÁREA DEL TRAPECIO  $\text{Área} = \frac{B + b}{2} \times h$ $\text{Perímetro} = B + b + L + l$ $\text{Área} = \frac{8 + 6}{2} \times 4 = 28 \text{ cm}^2$ $\text{Perímetro} = 8 + 6 + 5 + 5 = 24 \text{ cm}$	ÁREA DEL PARALELOGRAMO  $\text{Área} = B \times h$ $\text{Perímetro} = 2 \times (B + a)$ $\text{Área} = 8 \times 6 = 48 \text{ cm}^2$ $\text{Perímetro} = 2 \times (8 + 7) = 30 \text{ cm}$

www.Proferecursos.com

Fuente de la imagen: <https://www.proferecursos.com/areas-y-perimetros/>

3. Una función que reciba (pida por teclado) una cadena de texto de una ecuación de segundo orden y que halle las raíces usando la fórmula general. Por ejemplo: " $x^2+8x+15=0$ " o " $8x+15+x^2=0$ " voy a obtener $x_1 = -3$ y $x_2 = -5$.
4. Declarar un arreglo de 10 posiciones y con inicialización directa (no usar para este taller ciclos) llenarla con números aleatorios enteros entre -10 y 10 y si el número aleatorio es par se agregará cero en su lugar. Si el número es impar y es múltiplo de 3 entonces se guardará como 999 en su lugar. Para esto último, se debe declarar una función que genere un aleatorio y retorne el valor correspondiente según las restricciones dadas.
5. Crear una función que reciba 4 números como parámetros y retorne: valor mínimo, valor máximo, promedio, varianza y desviación estándar.
6. Crear una función que reciba 4 números como parámetros y me retorne un arreglo con los datos ordenados. (no se pueden usar ciclos).