Usu de variables y datos

Introducción

En la sección de uso de variables y datos en Go, se cubren temas fundamentales como:

- Declaración de variables y constantes: se muestra cómo declarar variables y constantes y cómo asignarles valores.
- Tipos de datos básicos: se describen los tipos de datos básicos en Go, como enteros, flotantes, booleanos y cadenas.
- Valores predeterminados: se explica qué valores se asignan a las variables y constantes cuando se declaran sin un valor explícito.
- Conversiones de tipos: se presentan varios métodos para convertir entre diferentes tipos de datos, incluyendo conversiones explícitas e implícitas.
- El paquete fmt: se introduce el paquete fmt, que proporciona funciones para imprimir y leer datos de entrada/salida.
- Operadores aritméticos y paquete math: se cubren los operadores aritméticos básicos y se introduce el paquete math para realizar operaciones matemáticas más complejas.
- Ejercicio: se proporciona un ejemplo práctico en el que se utilizan los conceptos anteriores para calcular e imprimir el área y el perímetro de un triángulo.

Al comprender estos conceptos, los programadores pueden comenzar a escribir programas simples y comprender mejor cómo trabajar con variables y datos en Go.

Declaración de variables

Vamos a comenzar este módulo examinando cómo se declaran y cómo se usan las variables en Go. Hay varias maneras de declarar una variable. Veremos cada una de ellas y escogerá la que mejor se adapte a sus necesidades o a su estilo. A medida que exploremos los conceptos básicos de las variables, destacaremos algunas características específicas de Go que no suelen presentar otros lenguajes de programación.

Para declarar una variable en Go, se utiliza la sintaxis `var nombreDeVariable tipoDeDato`. Por ejemplo, para declarar una variable de tipo cadena (string) llamada "firstName", se utilizaría el siguiente código:

```
var firstName string
```

Si quiere declarar otra variable, simplemente agregue una instrucción similar a la anterior. Puede declarar más de una variable en una sola línea si son del mismo tipo:

```
var firstName, lastName string
var age int
```

Otra forma de escribir la instrucción anterior es usar paréntesis después de la palabra clave var, como si tuviera un bloque dedicado para declarar variables:

```
var (
    firstName, lastName string
    age int
)
```

Variables inicializadas

También es posible inicializar una variable al mismo tiempo que se declara, utilizando la sintaxis `var nombreDeVariable tipoDeDato = valor`. Por ejemplo, para declarar e inicializar una variable de tipo cadena (string) llamada "nombre" con el valor "Juan", se utilizaría el siguiente código:

```
var (
    firstName string = "Alex"
    lastName  string = "Roel"
    age       int    = 27
)
```

Si decide inicializar una variable, no es necesario que especifique su tipo porque Go lo infiere al inicializar la variable con un valor. Por ejemplo, puede declarar e inicializar variables de esta manera:

```
var (
    firstName = "Alex"
    lastName  = "Roel"
    age       = 27
)
```

En Go, puede declarar e inicializar las variables en una sola línea. Separe cada nombre de variable con una coma y haga lo mismo para cada valor (en el mismo orden), como se muestra a continuación:

```
var (
    firstName, lastName, age = "Alex", "Roel", 27
)
```

Declaraciones de variables cortas

Go también cuenta con la sintaxis de declaraciones de variables cortas `nombreDeVariable := valor`, la cual permite declarar e inicializar una variable en la misma línea de código. Go deduce automáticamente el tipo de dato basado en el valor asignado a la variable. Por ejemplo, para declarar e inicializar una variable de tipo flotante (float32) llamada "precio" con el valor 2.50, se utilizaría el siguiente código:

```
package main

import "fmt"

func main() {
    firstName, lastName := "Alex", "Roel"
    age := 27
    fmt.Println(firstName, lastName, age)
}
```

Declaración de constantes

En Go, las constantes son variables cuyo valor no cambia durante la ejecución del programa. Se definen utilizando la palabra clave `const` seguida de un identificador y un valor. Por ejemplo:

```
const pi float32 = 3.14
const pi2 = 3.141592
```

En este ejemplo, `pi` es una constante que se utiliza para representar el valor de Pi, con un valor de `3.14159`.

Es importante tener en cuenta que, a diferencia de las variables normales, las constantes no se pueden modificar una vez que se han definido. Intentar cambiar el valor de una constante generará un error de compilación.

Las constantes pueden ser de cualquier tipo de dato básico, incluyendo enteros, flotantes, booleanos y cadenas. También se pueden definir constantes numéricas en diferentes bases, como octal y hexadecimal.

```
const (
    x = 100
    y = 0b1010  // binario
    z = 0o12    // octal
    w = 0xFF    // hexadecimal
)
```

En este ejemplo, se definen varias constantes numéricas en diferentes bases.

En resumen, las constantes en Go son variables cuyo valor no cambia y se definen utilizando la palabra clave `const`. Son útiles para definir valores que no deben ser modificados durante la ejecución del programa.

Tipos de datos básicos

En Go, existen diferentes tipos de datos que se pueden utilizar para almacenar diferentes tipos de valores. A continuación, se describen algunos de los tipos de datos más comunes en Go:

En Go hay cuatro categorías de tipos de datos:

- Tipos básicos: números, cadenas y booleanos
- Tipos agregados: matrices y estructuras
- Tipos de referencia: punteros, segmentos, mapas, funciones y canales
- Tipos de interfaz: interfaz

Tipos de datos numéricos

- `int` : representa valores enteros, puede ser de diferentes tamaños (`int8` , `int16` , `int32` , `int64`)
- `uint` : representa valores enteros positivos sin signo, también puede ser de diferentes tamaños (`uint8` , `uint16` , `uint32` , `uint64`)
- `float32` , `float64` : representan números decimales con diferentes niveles de precisión

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var integer8 int8 = 127
    var integer16 int16 = 32767
    var integer32 int32 = 2147483647
    var integer64 int64 = 9223372036854775807

    fmt.Println(integer8, integer16, integer32, integer64)
    /* Utiliza los constantes del paquete math
    para saber cantidad de números que puede almacenar*/
    fmt.Println(math.MaxInt8, math.MaxInt8)

    // Ejemplo con Uint
    var integerUint8 uint8 = 255
    fmt.Println(integerUint8, math.MaxUint8)

    //Ejemplo de tipo flotantes
    var numberFloat32 float32 = 2.54
    fmt.Println(numberFloat32, math.MaxFloat32)
}
```

Tipo de datos booleano

- `bool` : representa valores booleanos (verdadero o falso)

```
//Ejemplo de Booleano
var valueBool bool = true
```

Tipo de datos de texto

- `string` : representa una cadena de caracteres

A veces necesitará usar caracteres de escape. Para incluirlos en Go, use una barra diagonal inversa `(\)` antes del carácter. A continuación se muestran los ejemplos más comunes del uso de caracteres de escape:

- `\n` para líneas nuevas
- `\r` para retornos de carro
- `\t` para pestañas
- `\'` para comillas simples
- `\"` para comillas dobles
- `\\` para barras diagonales inversas

```
fullName := "Alex Roel \t(alias \"roelcode\")\n"
fmt.Println(fullName)
```

Byte y Rune

En Go, el tipo de datos "byte" es un alias para el tipo de datos "uint8". Un byte es un número entero sin signo que puede representar valores enteros en el rango de 0 a 255. El tipo de datos byte se utiliza comúnmente para representar datos de caracteres ASCII y otros datos que se almacenan en formato de byte.

```
func main() {
    var a byte = 'a' // Asignar el valor decimal 97 (equivalente a "a" en ASCII)
    fmt.Println(a)   // Imprimir el valor de la variable b

    s := "hola"       // Crear una cadena de texto
    fmt.Println(s[0]) // Imprimir el valor del primer byte de la cadena s
}
```

Por otro lado, el tipo de datos "rune" es un alias para el tipo de datos "int32". Rune representa un punto de código Unicode y se utiliza para representar caracteres Unicode en Go. En Go, los caracteres Unicode se codifican utilizando UTF-8, que es un formato de codificación de caracteres que utiliza uno o más bytes para representar cada carácter Unicode. La palabra "rune" se utiliza para enfatizar que estos valores representan caracteres y no simplemente valores enteros.

```
func main() {
    var r rune = '❤' // Asignar el valor de un caracter Unicode
    fmt.Println(r)   // Imprimir el valor de la variable r
}
```

En este ejemplo, se declara una variable "r" de tipo "rune" y se le asigna el valor de un caracter Unicode en su representación de 32 bits. Luego, se imprime el valor de la variable "r" en la consola.

Valores predeterminados

En Go, todas las variables tienen un valor predeterminado, conocido como valor cero. El valor cero depende del tipo de dato de la variable y se utiliza cuando no se ha asignado un valor explícitamente a la variable.

A continuación, se muestran los valores cero de algunos de los tipos de datos más comunes en Go:

- `int` : `0`
- `float` : `0.0`
- `bool` : `false`
- `string` : cadena vacía (`""`)
- `array` : un array vacío
- `slice` : un slice nulo (`nil`)
- `map` : un map nulo (`nil`)
- `struct` : todas las variables dentro de la estructura tienen su propio valor cero

```
var defaultInt int
var defaultUint uint
var defaultFloat64 float64
var defaultBool bool
var defaultString string
fmt.Println(defaultInt, defaultUint, defaultFloat64, defaultBool, defaultString)
```

Es importante tener en cuenta que el valor cero no siempre es el mismo que el valor nulo (`nil`). Por ejemplo, el valor cero de un slice es un slice nulo, no un slice vacío. Es importante comprender la diferencia para evitar errores en la programación.

En resumen, es importante conocer los valores cero de los diferentes tipos de datos en Go para comprender cómo funcionan las variables en diferentes contextos y evitar errores al programar.

Conversiones de tipos

En Go, la conversión debe llevarse a cabo explícitamente. Este lenguaje de programación ofrece algunas maneras nativas de convertir un tipo de datos en otro diferente. Por ejemplo, una de ellas consiste en usar la función integrada para cada tipo de la siguiente forma:

```
func main() {
    var integer16 int16 = 50
    var integer32 int32 = 100
    fmt.Println(int32(integer16) + integer32)
}
```

Sí, otra forma de realizar conversiones de tipos en Go es utilizando el paquete `strconv` . Este paquete proporciona funciones para convertir valores entre diferentes tipos de datos, incluidos enteros, flotantes, booleanos y cadenas.

El paquete `strconv` se utiliza para convertir valores numéricos y de cadena, y ofrece una serie de funciones para realizar diferentes tipos de conversiones.

```
package main

import (
    "fmt"
    "strconv"
)

func main() {
    // Convertir una cadena a un número entero
    s := "100"
    i, _ := strconv.Atoi(s)
    fmt.Println(i + i)

    // Convertir un número entero a una cadena
    n := 42
    s = strconv.Itoa(n)
    fmt.Println(s + s)
}
```

El paquete fmt

El paquete fmt es uno de los paquetes más utilizados en Go. Proporciona funciones para imprimir y formatear la salida en la consola, así como para leer la entrada del usuario. Algunas de las funciones más comunes en este paquete incluyen:

- `fmt.Print` : Imprime los argumentos en la consola sin agregar un carácter de nueva línea al final.
- `fmt.Println` : Imprime los argumentos en la consola y agrega un carácter de nueva línea al final.
- `fmt.Printf` : Imprime los argumentos formateados en la consola, de acuerdo con un formato especificado.
- `fmt.Scanf` : Lee la entrada del usuario desde la consola y la almacena en variables proporcionadas.
- `fmt.Scanln` : Es una función de entrada que permite al usuario ingresar datos desde la consola y almacenarlos en variables. La función espera una lista de argumentos de puntero, que se utilizarán para almacenar los datos ingresados por el usuario.
- `fmt.Sprintf` : La función `fmt.Sprintf` es una variante de la función `fmt.Printf` que devuelve una cadena en lugar de escribir los datos en la salida estándar.

```
func main() {
    name := "ALex"
    age := 27

    fmt.Printf("Hola, me llamo %s y tengo %d años.\n", name, age)

    greeting := fmt.Sprintf("Hola, me llamo %s y tengo %d años.", name, age)
    fmt.Println(greeting)
}
```

En Go, también se puede utilizar el verbo de formato %T de la función fmt.Printf para obtener el tipo de una variable. Este verbo de formato imprime el tipo de la variable, en lugar de su valor.

```
fmt.Printf("El tipo de name es: %T\n", name)
fmt.Printf("El tipo de age es: %T\n", name)
```

Operadores aritméticos y Paquete Math

En Go, hay varios tipos de operadores matemáticos que se utilizan para realizar operaciones aritméticas, incrementar o decrementar valores, y asignar valores a variables. A continuación se describen los operadores más comunes:

Operadores aritméticos:

- `+` : suma dos valores.
- `-` : resta dos valores.
- `*` : multiplica dos valores.
- `/` : divide el primer valor por el segundo valor.

- `%` : devuelve el resto de la división entre dos valores (módulo).

```
a := 10
b := 3
fmt.Println(a + b)  // 13
fmt.Println(a - b)  // 7
fmt.Println(a * b)  // 30
fmt.Println(a / b)  // 3
fmt.Println(a % b)  // 1
```

Operadores de incremento y decremento:

- `++` : incrementa el valor de una variable en 1.
- `--` : decrementa el valor de una variable en 1.

```
a := 10
a++
fmt.Println(a)  // 11

b := 5
b--
fmt.Println(b)  // 4
```

Operadores en asignación:

- `=` : asigna el valor de la derecha a la variable de la izquierda.
- `+=` : suma el valor de la derecha a la variable de la izquierda y luego asigna el resultado a la variable de la izquierda.
- `-=` : resta el valor de la derecha a la variable de la izquierda y luego asigna el resultado a la variable de la izquierda.
- `*=` : multiplica el valor de la derecha a la variable de la izquierda y luego asigna el resultado a la variable de la izquierda.
- `/=` : divide el valor de la variable de la izquierda por el valor de la derecha y luego asigna el resultado a la variable de la izquierda.
- `%=` : devuelve el resto de la división entre la variable de la izquierda y la variable de la derecha y luego asigna el resultado a la variable de la izquierda.

```
a := 10
a += 5
fmt.Println(a)  // 15

b := 20
b -= 3
fmt.Println(b)  // 17

c := 7
c *= 3
fmt.Println(c)  // 21

d := 100
d /= 5
fmt.Println(d)  // 20

e := 15
e %= 4
fmt.Println(e)  // 3
```

En resumen, los operadores matemáticos en Go se utilizan para realizar operaciones aritméticas, incrementar o decrementar valores, y asignar valores a variables. Es importante conocer los diferentes operadores y cómo se utilizan para realizar operaciones matemáticas y asignar valores a variables en Go.

Paquetes Math

El paquete math en Go proporciona funciones matemáticas comunes que se utilizan para realizar cálculos matemáticos complejos. Este paquete incluye funciones para trabajar con valores numéricos, como operaciones trigonométricas, logaritmos, exponenciales, funciones de redondeo y más.

Para utilizar el paquete math, simplemente se debe importar en el archivo Go y luego llamar a la función apropiada. A continuación se muestra un ejemplo de algunas de las funciones disponibles en el paquete math:

Constantes en Math

El paquete math no proporciona constantes predefinidas, pero sí tiene varias funciones que retornan constantes matemáticas, por ejemplo:

- `math.Pi` : retorna el valor de Pi (π).
- `math.E` : retorna el valor de la constante matemática e. Estas constantes se pueden utilizar en cualquier programa que utilice el paquete math. Aquí un ejemplo:ç

```
fmt.Println("PI: ", math.Pi) //PI:  3.141592653589793
fmt.Println("E: ", math.E)   //E:  2.718281828459045
```

Funciones de potencia

El paquete math proporciona varias funciones para calcular potencias de un número en Go. A continuación se muestran las funciones de potencia disponibles:

- `math.Pow(x, y float64) float64` : devuelve x elevado a la potencia de y.
- `math.Sqrt(x float64) float64` : devuelve la raíz cuadrada de x.
- `math.Cbrt(x float64) float64` : devuelve la raíz cúbica de x.
- `math.Pow10(n int) float64` : devuelve 10 elevado a la potencia de n.

```
// Ejemplo de potencias
fmt.Println(math.Pow(2, 3)) // Imprime: 8
fmt.Println(math.Sqrt(64)) // Imprime: 8
fmt.Println(math.Cbrt(27)) // Imprime: 3
fmt.Println(math.Pow10(2)) // Imprime: 100
```

En resumen, el paquete math es muy útil para realizar operaciones matemáticas en Go, ya que proporciona una gran cantidad de funciones matemáticas estándar que se pueden utilizar en cualquier programa com:

- Funciones de redondear un número
- Funciones trigonométricas
- Funciones exponenciales y logarítmicas

Ejercicio: Calcule e imprima el área y el perímetro del triángulo

Crear un programa que solicite al usuario que ingrese los lados de un triángulo rectángulo y luego calcule e imprima el área y el perímetro del triángulo.

El programa debe:

- Solicitar al usuario que ingrese la longitud de los dos lados del triángulo rectángulo.
- Calcular la hipotenusa del triángulo usando el teorema de Pitágoras.
- Calcular el área del triángulo usando la fórmula base x altura / 2.
- Calcular el perímetro del triángulo sumando los lados.
- Imprimir el área y el perímetro del triángulo con dos decimales de precisión.
- El programa debe usar variables para almacenar los lados del triángulo, la hipotenusa, el área y el perímetro. También debe usar constantes para representar el número de decimales de precisión que se desean en la salida.

Además, se deben utilizar funciones del paquete Math de Go para calcular la raíz cuadrada y cualquier otro cálculo matemático necesario.

Ejemplo de entrada y salida:

```
Ingrese lado 1: 3.5
Ingrese lado 2: 4.2

Área: 7.35
Perímetro: 12.20
```

Por supuesto, aquí te presento el código para crear un programa en Go que calcule el área y el perímetro de un triángulo rectángulo:

```
package main

import (
    "fmt"
    "math"
)

func main() {
    var lado1, lado2 float64
    const precision = 2

    fmt.Print("Ingrese lado 1: ")
    fmt.Scanln(&lado1)

    fmt.Print("Ingrese lado 2: ")
    fmt.Scanln(&lado2)

    hipotenusa := math.Sqrt(math.Pow(lado1, 2) + math.Pow(lado2, 2))
    area := (lado1 * lado2) / 2
    perimetro := lado1 + lado2 + hipotenusa

    fmt.Printf("\nÁrea: %.*f\n", precision, area)
    fmt.Printf("Perímetro: %.*f\n", precision, perimetro)
}
```

Para imprimir los resultados, usamos la función fmt.Printf() para formatear la salida y mostrar los valores calculados. La cadena de formato "%.*f" se utiliza para indicar que se mostrará un número de punto flotante con un número determinado de decimales (especificado por la constante precision), y el operador % se utiliza para indicar qué valor se utilizará en ese lugar de la cadena de formato.

Resumen

En la sección de uso de variables y datos en Go, se cubrieron varios temas fundamentales para trabajar con variables y datos en este lenguaje de programación.

En primer lugar, se explicó cómo declarar variables y constantes en Go y cómo asignarles valores. Se presentaron los tipos de datos básicos, incluyendo enteros, flotantes, booleanos y cadenas, y se explicó cómo se utilizan en Go.

También se habló de los valores predeterminados que se asignan a las variables y constantes cuando se declaran sin un valor explícito, y se cubrieron los diferentes métodos para convertir entre diferentes tipos de datos.

Además, se presentó el paquete fmt, que proporciona funciones para imprimir y leer datos de entrada/salida, y se describieron los operadores aritméticos básicos y el paquete math para realizar operaciones matemáticas más complejas.

Por último, se proporcionó un ejemplo práctico en el que se utilizaron los conceptos anteriores para calcular e imprimir el área y el perímetro de un triángulo.

Al comprender estos conceptos, los programadores pueden comenzar a escribir programas simples y comprender mejor cómo trabajar con variables y datos en Go.