Part 6: Functions

01 APRIL 2017

This is the sixth tutorial in Golang tutorial series.

A function is a block of code that performs a specific task. A function takes a input, performs some calculations on the input and generates a output.

Function declaration

The general syntax for declaring a function in go is

```
func functionname(parametername type) returntype {
 //function body
}
```

The function declaration starts with a `func` keyword followed by the `functionname`. The parameters are specified between `(` and `)` followed by the `returntype` of the function. The syntax for specifying a parameter is parameter name followed by the type. Any number of parameters can be specified like `(parameter1 type, parameter2 type)`. Then there is a block of code between `{` and `}` which forms the body of the function.

The parameters and return type are optional in a function. Hence the following syntax is also a valid function declaration.

```
func functionname() {
}
```

Sample Function

Lets write a function which takes the price of a single product and number of products as input parameters and calculates the total price by multiplying these two values and returns the output.

```
func calculateBill(price int, no int) int {
    var totalPrice = price * no
    return totalPrice
}
```

The above function has two input parameters `price` and `no` of type int and it returns the `totalPrice` which is the product of price and no. The return value is also of type int.

If consecutive parameters are of the same type, we can avoid writing the type each time and it is enough to be written once at the end.ie `price int, no int` can be written as `price, no int`. The above function can hence be rewritten as,

```
func calculateBill(price, no int) int {
    var totalPrice = price * no
    return totalPrice
}
```

Now that we have a function ready, lets call it from somewhere in the code. The syntax for calling a function is `functionname(parameters)`. The above function can be called using the code.

```
calculateBill(10, 5)
```

Here is the complete program which uses the above function and prints the total price.

```
package main

import (
    "fmt"
)

func calculateBill(price, no int) int {
    var totalPrice = price * no
    return totalPrice
}
func main() {
    price, no := 90, 6
    totalPrice := calculateBill(price, no)
    fmt.Println("Total price is ", totalPrice)
}
```

Multiple return values

It is possible to return multiple values from a function. Lets write a function `rectProps` which takes the length and width of a rectangle and returns both the area and perimeter of the rectangle. The area of the rectangle is the product of length and width and the perimeter is twice the sum of the length and width.

```
package main

import (
    "fmt"
)

func rectProps(length, width float64)(float64, float64) {
    var area = length * width
    var perimeter = (length + width) * 2
    return area, perimeter
}

func main() {
    area, perimeter := rectProps(10.8, 5.6)
    fmt.Printf("Area %f Perimeter %f", area, perimeter)
}
```

If a function returns multiple return values then they should be specified between `(` and `)`. `func rectProps(length, width float64)(float64, float64)` has two float64 parameters(length and width) and also returns two float64 values. The above program outputs `Area 60.480000 Perimeter 32.800000`

Named return values

It is possible to return named values from a function. If a return value is named, it can be considered as being declared as a variable in the first line of the function.

The above rectProps can be rewritten using named return values as

```
func rectProps(length, width float64)(area, perimeter f
    area = length * width
    perimeter = (length + width) * 2
    return //no explicit return value
}
```

area and **perimeter** are the named return values in the above function. Note that the return statement in the function does not explicitly return any value. Since `area` and `perimeter` are specified in the function declaration as return values, they are automatically returned from the function when a return statement in encountered.

Blank Identifier

_ is know as the blank identifier in Go. It can be used in place of any value of any type. Lets see what's the use of this blank identifier.

The `rectProps` function returns the area and perimeter of the rectangle. What if we only need the `area` and want to discard the `perimeter`. This is where _ is of use.

The program below uses only the `area` returned from the `rectProps` function.

```
package main

import (
    "fmt"
)

func rectProps(length, width float64) (float64, float64
    var area = length * width
    var perimeter = (length + width) * 2
    return area, perimeter
}
func main() {
    area, _ := rectProps(10.8, 5.6) // perimeter is disc
    fmt.Printf("Area %f ", area)
}
```

In the line `area, _ := rectProps(10.8, 5.6)` we use only the area and the _ identifier is used to discard the parameter.

Thats it for functions. Please leave your valuable comments and feedback. Thanks for reading.

Next tutorial - Packages

Naveen Ramanathan

iOS developer at dietco.de and Golang enthusiast.

Share this post

About

For any queries/suggestions, please contact us at **naveen[at]golangbot[dot]com**

Follow Us

Newsletter

Join Our Newsletter

Signup for our newsletter and get the **Golang tools cheat sheet for free**

Email*

Subscribe