Part 3: Variables

11 FEBRUARY 2017

This is the third tutorial in our Golang tutorial series and it deals with variables in golang.

You can read the Golang tutorial part 2: Hello World to learn about configuring golang and running the hello world program.

What is a variable

Variable is the name given to a memory location to store a value of a specific type. There are various syntaxes to declare variables in go.

Declaring a single variable

var name type is the syntax to declare a single variable.

```
package main

import "fmt"

func main() {
    var age int // variable declaration
    fmt.Println("my age is", age)
}
```

The statement `var age int` declares a variable named *age* of type *int*. We have not assigned any value for the variable. If a variable is not assigned any value, go automatically initialises it with the **zero value** of the variable's type. In this case age is assigned the value 0. If you run this program, you can see the following output

```
my age is 0
```

A variable can be assigned to any value of its type. In the above program age can be assigned any integer value.

```
package main

import "fmt"

func main() {
    var age int // variable declaration
    fmt.Println("my age is ", age)
    age = 29 //assignment
    fmt.Println("my age is", age)
    age = 54 //assignment
    fmt.Println("my new age is", age)
}
```

The above program will produce the following output.

```
my age is  0
my age is 29
my new age is 54
```

Declaring a variable with initial value

A variable can also be given a initial value when it is declared.

var name type = initialvalue is the syntax to declare a variable with initial value.

```
package main

import "fmt"

func main() {
    var age int = 29 // variable declaration with initia

    fmt.Println("my age is", age)
}
```

In the above program, age is variable of type int and has initial value 29. If you run the above program, you can see the following output. It shows that age has been initialised with the value 29.

```
my age is 29
```

Type inference

If a variable has an initial value, Go will automatically be able to infer the type of that variable using that initial value. Hence if a variable has an initial value, the *type* in the variable declaration can be omitted.

If the variable is declared using the syntax **var name = initialvalue**, Go will automatically infer the type of that variable from the initial value.

In the following example, you can see that the type *int* of the variable *age* has been removed. Since the variable has a initial value of 29, go can infer that its of type int.

```
package main

import "fmt"

func main() {
    var age = 29 // type will be inferred

    fmt.Println("my age is", age)
}
```

Multiple variable declaration

Multiple variables can be declared in a single statement.

var name1, name2 type = initialvalue1, initialvalue2 is the syntax for multiple variable declaration.

```
package main

import "fmt"

func main() {
    var width, height int = 100, 50 //declaring multiple

    fmt.Println("width is", width, "height is", height)
}
```

The *type* can be omitted if the variables have initial value. The program below declares multiple variables using type inference.

```
package main

import "fmt"

func main() {
    var width, height = 100, 50 //"int" is dropped

    fmt.Println("width is", width, "height is", height)
}
```

If you run the above program, you can see `width is 100 height is 50` printed as the output.

As you would have probably guessed by now, if the initial value is ignored for width and height, they will have 0 assigned as their initial value.

```
package main

import "fmt"

func main() {
    var width, height int
    fmt.Println("width is", width, "height is", height)
    width = 100
    height = 50
    fmt.Println("new width is", width, "new height is ",
}
```

The above program will output

```
width is 0 height is 0
new width is 100 new height is  50
```

There might be cases where we would want to declare variables belonging to different types in a single statement. The syntax for doing that is

```
var (
      name1 = initialvalue1,
      name2 = initialvalue2
)
```

The following program uses the above syntax to declare variables of different types.

```
package main

import "fmt"

func main() {
    var (
        name   = "naveen"
        age    = 29
        height int
    )
    fmt.Println("my name is", name, ", age is", age, "ar
}
```

Here we declare a variable **name of type string, age and height of type int**. (We will discuss about the various types available in golang in the next tutorial). Running the above program will produce the output `my name is naveen , age is 29 and height is 0`

Short hand declaration

Go also provides another concise way for declaring variables. This is known as short hand declaration and it uses := operator.

name := initialvalue is the short hand syntax to declare a variable.

```
package main

import "fmt"

func main() {
    name, age := "naveen", 29 //short hand declaration

    fmt.Println("my name is", name, "age is", age)
}
```

If you run the above program, you can see `my name is naveen age is 29` as the output.

Short hand declaration requires initial values for all variables in the left hand side of the assignment. The following program will thrown an error `assignment count mismatch: 2 = 1`. This is because **age has not been assigned a value.**

```
package main

import "fmt"

func main() {
    name, age := "naveen" //error

    fmt.Println("my name is", name, "age is", age)
}
```

Short hand syntax can only be used when at least one of the variables in the left side of := is newly declared. Consider the following program

```
package main

import "fmt"

func main() {
    a, b := 20, 30 // declare variables a and b
    fmt.Println("a is", a, "b is", b)
    b, c := 40, 50 // b is already declared but c is new
    fmt.Println("b is", b, "c is", c)
    b, c = 80, 90 // assign new values to already declar
    fmt.Println("changed b is", b, "c is", c)
}
```

In the above program, in line `b, c := 40, 50` **b** has already been declared but **c** is newly declared and hence it works and outputs

```
a is 20 b is 30
b is 40 c is 50
changed b is 80 c is 90
```

Whereas if we run the program below,

```
package main

import "fmt"

func main() {
    a, b := 20, 30 //a and b declared
    fmt.Println("a is", a, "b is", b)
    a, b := 40, 50 //error, no new variables
}
```

it will throw error `8: no new variables on left side of :=` This is because both the variables **a** and **b** have already been declared and there are no new variables in the left side of :=

Variables can also be assigned values which are computed during run time. Consider the following program.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    a, b := 145.8, 543.8
    c := math.Min(a, b)
    fmt.Println("minimum value is ", c)
}
```

In the above program, the value of c is calculated during run time and its the minimum of a and b.

Variables declared as belonging to one type cannot be assigned a value of another type. The following program will throw an error `cannot use "naveen" (type string) as type int in assignment` because age is declared as type int and we are trying to assign a string value to it.

```
package main

func main() {
    age := 29      // age is int
    age = "naveen" // error since we are trying to assi
}
```

Thanks for reading. The next part Golang tutorial part 4: Types of this series focuses on types in golang. Please post your feedback and queries in the comments section.

Naveen Ramanathan

iOS developer at dietco.de and Golang enthusiast.

Share this post