Go - Up and Running

https://github.com/BilalMaz/DevOps-Architect-BootCamp

About me

Hi there, my name is Bilal and I will Welcome you to DevOps boot camp! I am thrilled to have you join us for this exciting journey of learning and discovery.

In this boot camp, we will be exploring the principles and practices of DevOps, which is a set of methodologies and tools that aims to bridge the gap between software development and operations. DevOps is an increasingly important area in the field of software engineering, as it helps organizations to streamline their processes, improve their agility, and deliver better value to their customers.

By the end of this boot camp, you will have gained a comprehensive understanding of DevOps and its key concepts, as well as practical skills in areas such as infrastructure automation, continuous integration and delivery, monitoring and logging, and more. You will be equipped with the knowledge and tools to apply DevOps principles in your own work and contribute to the success of your organization.

I am always looking to connect with other professionals in the field, share ideas and insights, and stay up to date on the latest trends and developments. I welcome the opportunity to connect with you and explore ways in which we can collaborate and support each other.

Bilal Mazhar

Please find my Linkedin profile

https://www.linkedin.com/in/bilalmazhar-cyber-security-consultant/

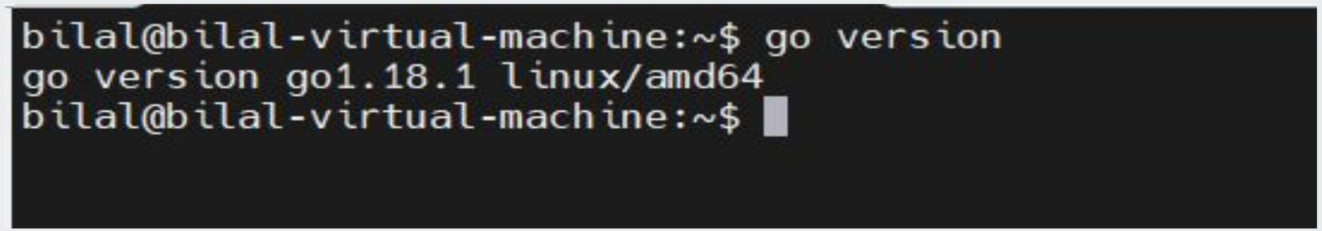
What is Go ?

Go is a compiled programming language. Before you run a program, Go uses a compiler to translate your code into the 1s and 0s that machines speak. It compiles all your code into a single executable for you to run or distribute. During this process, the Go compiler can catch typos and mistakes

Installation

→ apt get-install go

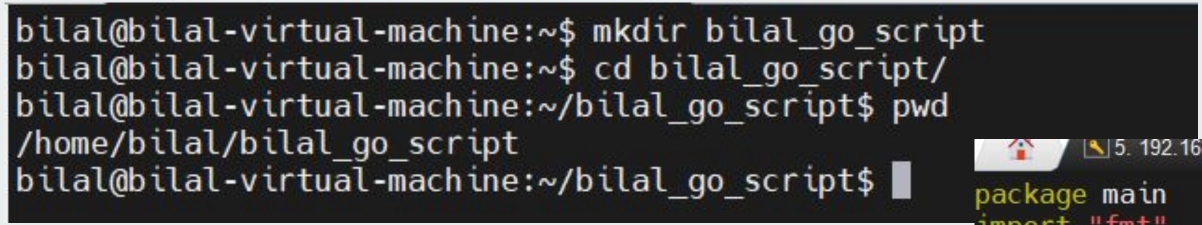
→ go version



```
bilal@bilal-virtual-machine:~$ go version
go version go1.18.1 linux/amd64
bilal@bilal-virtual-machine:~$
```

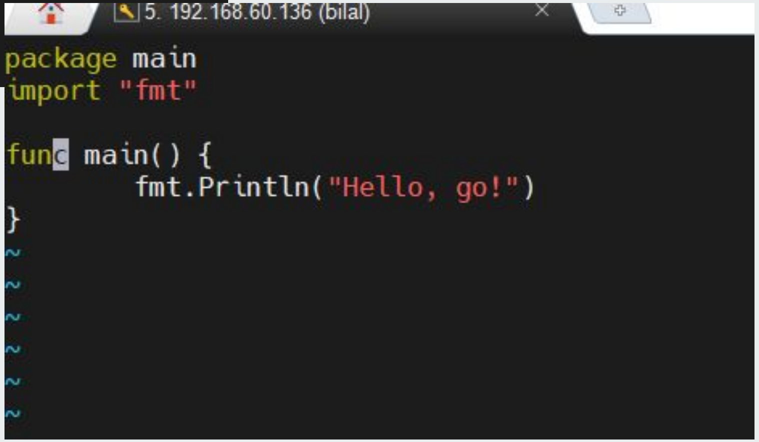
Hello world !

→ How to get started with Go? well, it is easy. Firstly, create a folder, called hello.



```
bilal@bilal-virtual-machine:~$ mkdir bilal_go_script
bilal@bilal-virtual-machine:~$ cd bilal_go_script/
bilal@bilal-virtual-machine:~/bilal_go_script$ pwd
/home/bilal/bilal_go_script
bilal@bilal-virtual-machine:~/bilal_go_script$ 
```

→ Create a file with go extension example : “**bilal_fisrt_script.go**” in new created directory



```
package main
import "fmt"

func main() {
        fmt.Println("Hello, go!")
}
~
~
~
~
~
~
```

Example explained

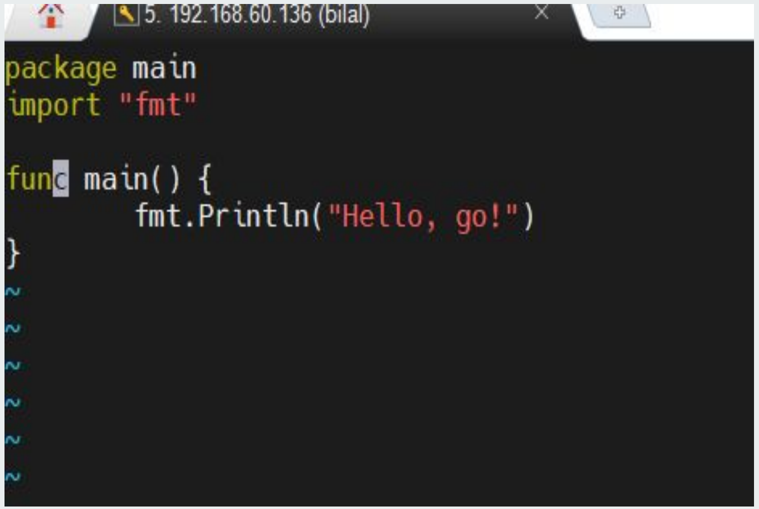
Line 1: In Go, every program is part of a package. We define this using the package keyword. In this example, the program belongs to the main package.

Line 2: import ("fmt") lets us import files included in the fmt package.

Line 3: A blank line. Go ignores white space. Having white spaces in code makes it more readable.

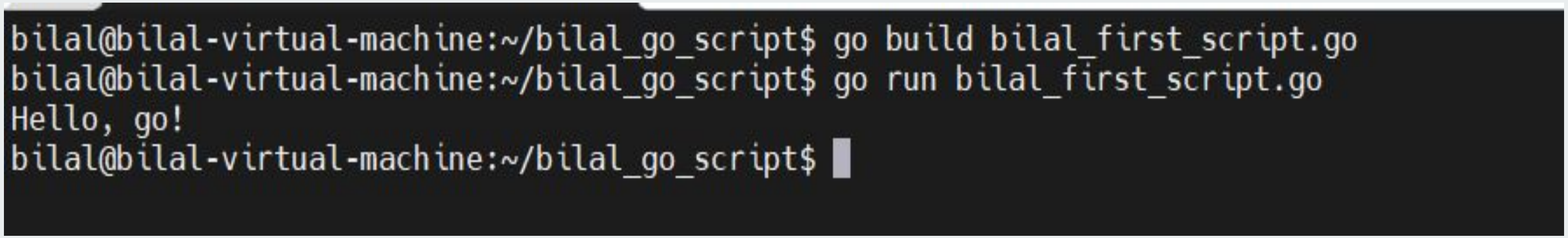
Line 4: func main() {} is a function. Any code inside its curly brackets {} will be executed.

Line 5: fmt.Println() is a function made available from the fmt package. It is used to output/print text. In our example it will output "**Hello World**!".



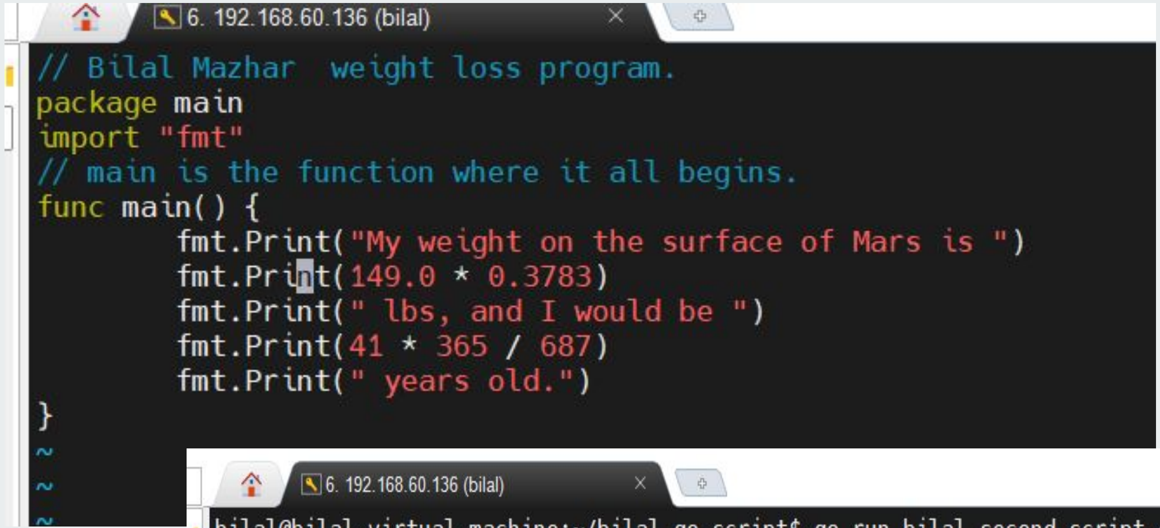
```
package main
import "fmt"

func main() {
        fmt.Println("Hello, go!")
}
~
~
~
~
~
~
```

→ Save this file. Then, back to your Terminal. You can build and run it.

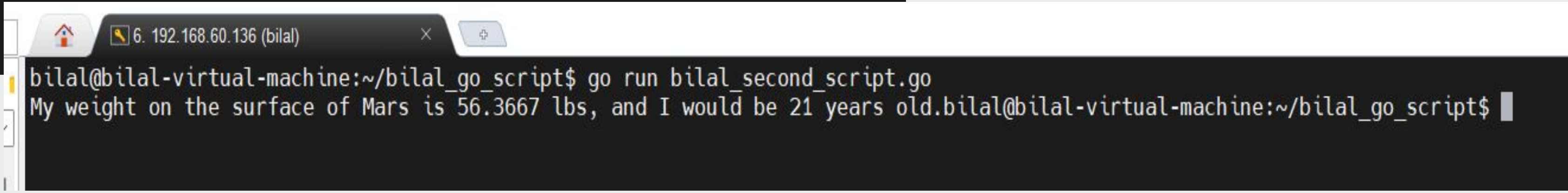


```
bilal@bilal-virtual-machine:~/bilal_go_script$ go build bilal_first_script.go
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_first_script.go
Hello, go!
bilal@bilal-virtual-machine:~/bilal_go_script$ 
```

Example 1



```
// Bilal Mazhar  weight loss program.
package main
import "fmt"
// main is the function where it all begins.
func main() {
        fmt.Print("My weight on the surface of Mars is ")
        fmt.Print(149.0 * 0.3783)
        fmt.Print(" lbs, and I would be ")
        fmt.Print(41 * 365 / 687)
        fmt.Print(" years old.")
}
~
~
~
```



```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_second_script.go
My weight on the surface of Mars is 56.3667 lbs, and I would be 21 years old.bilal@bilal-virtual-machine:~/bilal_go_script$
```

Go Variables

→ Variables are containers for storing data values.

→ In Go, there are different types of variables, for example:

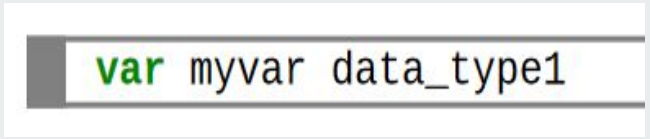
- **Int-** stores integers (whole numbers), such as 123 or -123
- **Float32-** stores floating point numbers, with decimals, such as 19.99 or -19.99
- **String** - stores text, such as "Hello World". String values are surrounded by double quotes
- **Bool-** stores values with two states: true or false

Declaring (Creating) Variables

In Go, there are two ways to declare a variable:

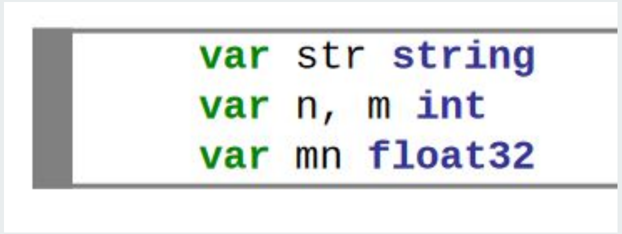
→ With the var keyword:

Use the var keyword, followed by variable name and type:



```
var myvar data_type1
```

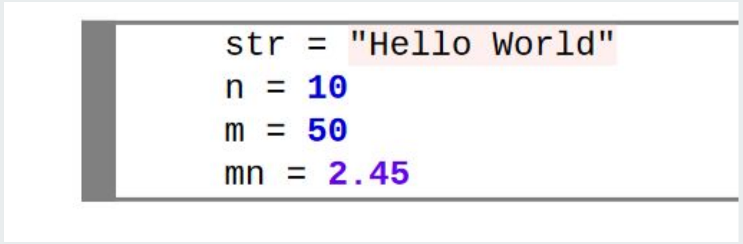
→ The following is a sample script to declare some variables.



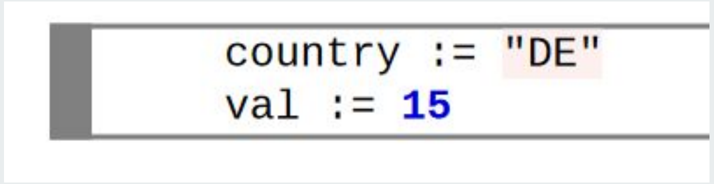
```
        var str string
        var n, m int
        var mn float32
```

Assigning Variables

Variable that you already declare can be assigned by a value. It can done using the equals sign (=). For example, variable str will assign a string value “**Hello World**”, you would write this:

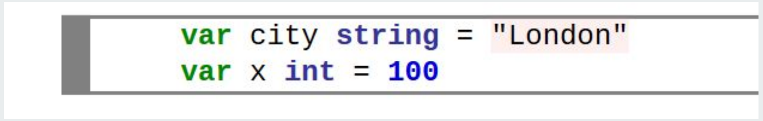


```
str = "Hello World"
n = 10
m = 50
mn = 2.45
```



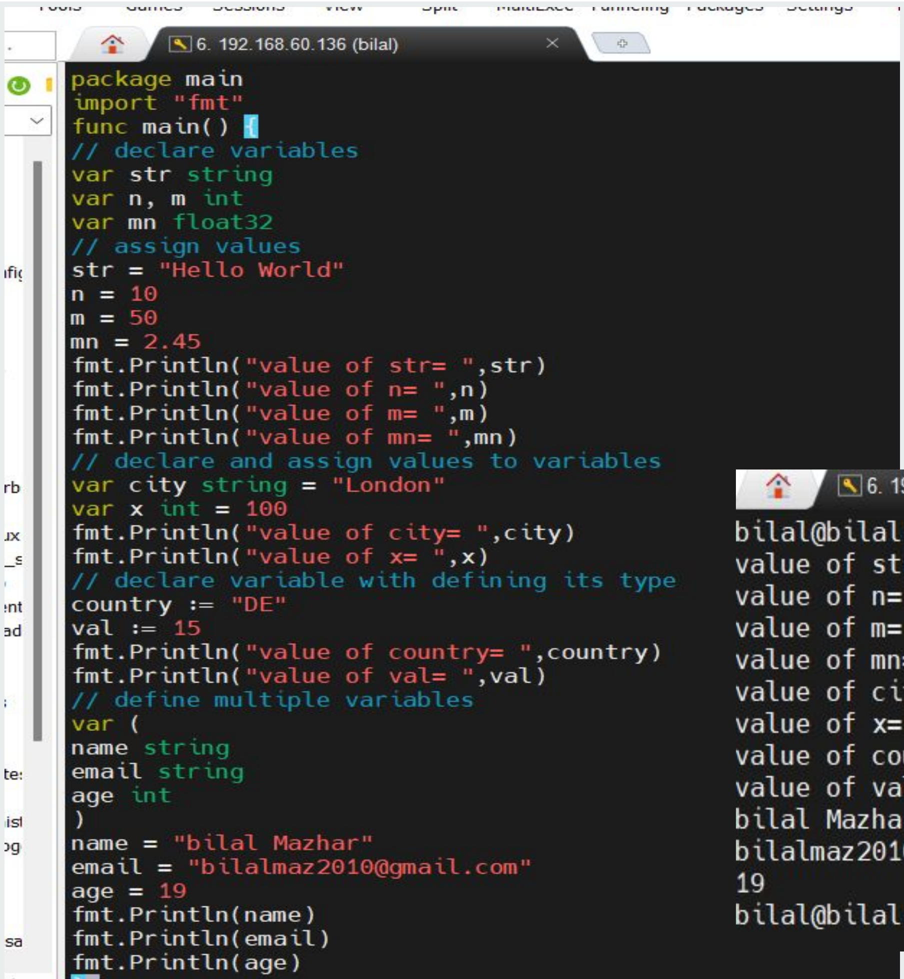
```
country := "DE"
val := 15
```

We also can declare a variable and assign its value.

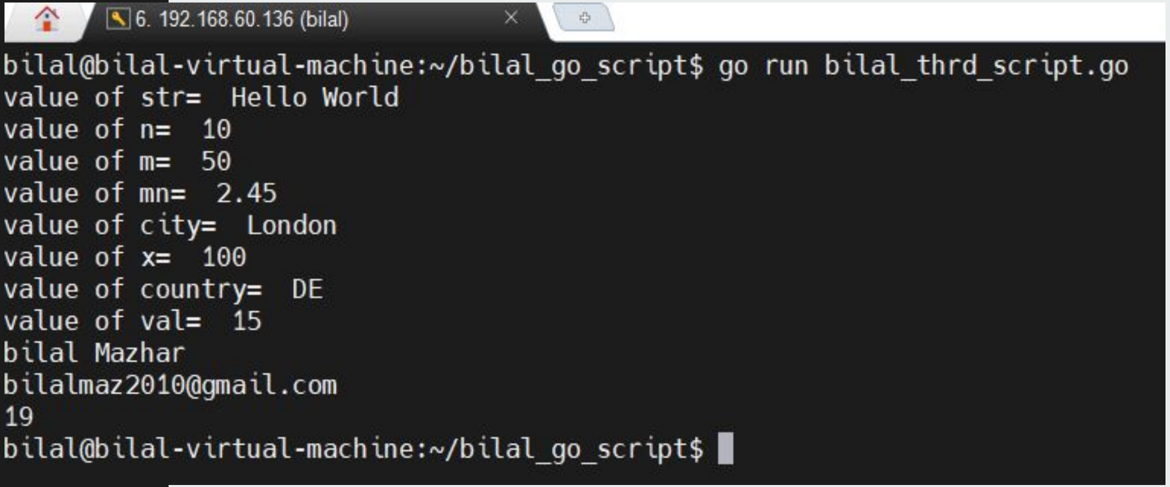


```
var city string = "London"
var x int = 100
```

Example 2

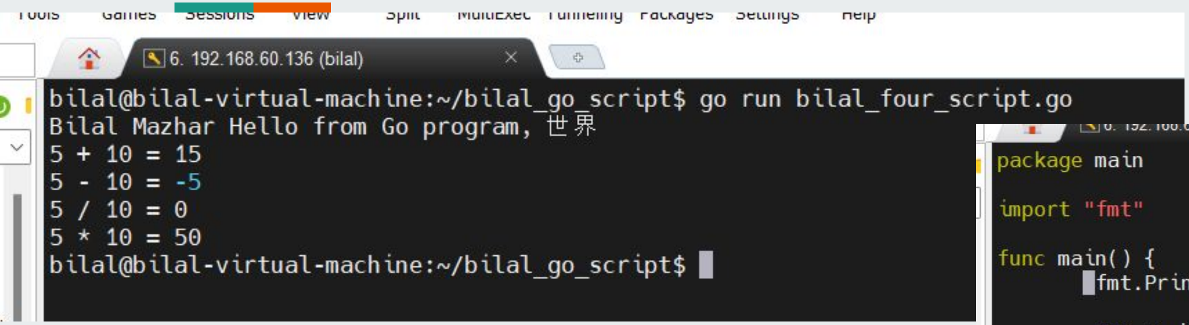


```
package main
import "fmt"
func main() {
// declare variables
var str string
var n, m int
var mn float32
// assign values
str = "Hello World"
n = 10
m = 50
mn = 2.45
fmt.Println("value of str= ",str)
fmt.Println("value of n= ",n)
fmt.Println("value of m= ",m)
fmt.Println("value of mn= ",mn)
// declare and assign values to variables
var city string = "London"
var x int = 100
fmt.Println("value of city= ",city)
fmt.Println("value of x= ",x)
// declare variable with defining its type
country := "DE"
val := 15
fmt.Println("value of country= ",country)
fmt.Println("value of val= ",val)
// define multiple variables
var (
name string
email string
age int
)
name = "bilal Mazhar"
email = "bilalmaz2010@gmail.com"
age = 19
fmt.Println(name)
fmt.Println(email)
fmt.Println(age)
```

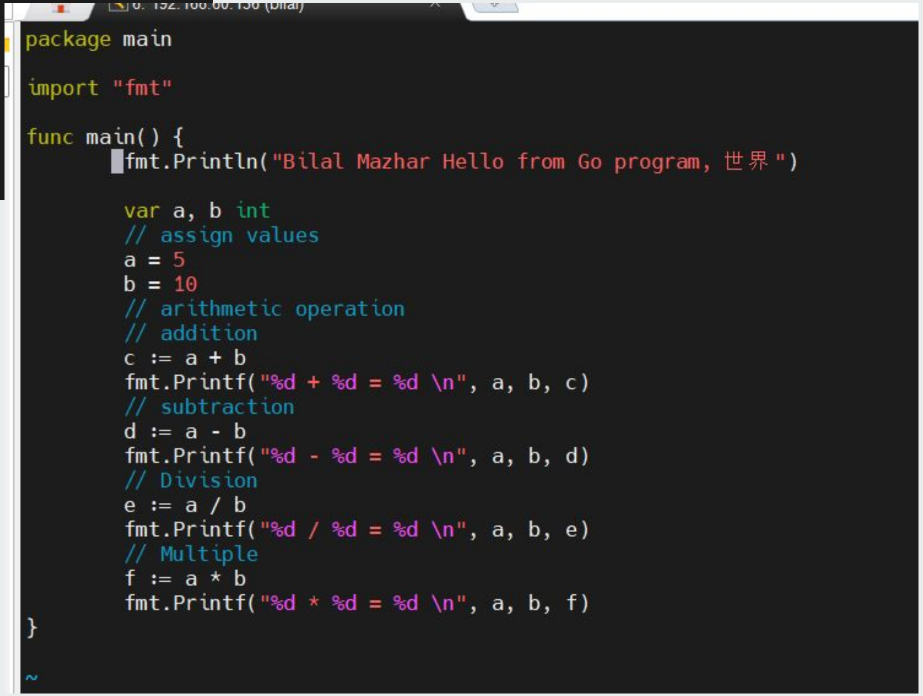


```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_thrd_script.go
value of str=  Hello World
value of n=  10
value of m=  50
value of mn=  2.45
value of city=  London
value of x=  100
value of country=  DE
value of val=  15
bilal Mazhar
bilalmaz2010@gmail.com
19
bilal@bilal-virtual-machine:~/bilal_go_script$
```

Example 3



```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_four_script.go
Bilal Mazhar Hello from Go program, 世界
5 + 10 = 15
5 - 10 = -5
5 / 10 = 0
5 * 10 = 50
bilal@bilal-virtual-machine:~/bilal_go_script$
```



```go
package main

import "fmt"

func main() {
        fmt.Println("Bilal Mazhar Hello from Go program, 世界")

        var a, b int
        // assign values
        a = 5
        b = 10
        // arithmetic operation
        // addition
        c := a + b
        fmt.Printf("%d + %d = %d \n", a, b, c)
        // subtraction
        d := a - b
        fmt.Printf("%d - %d = %d \n", a, b, d)
        // Division
        e := a / b
        fmt.Printf("%d / %d = %d \n", a, b, e)
        // Multiple
        f := a * b
        fmt.Printf("%d * %d = %d \n", a, b, f)
}

~
```

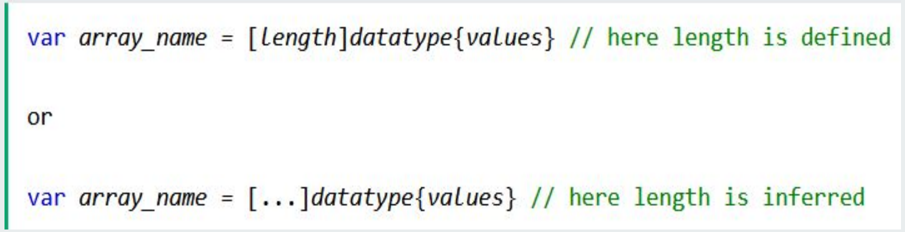
Go Arrays

→ Arrays are used to store multiple values of the same type in a single variable, instead of declaring separate variables for each value.

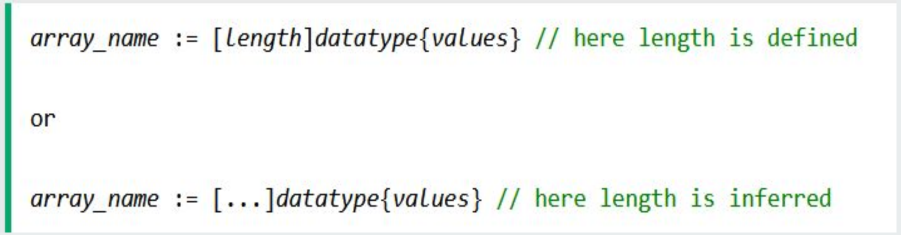
Declare an Array

In Go, there are two ways to declare an array:

1. With the var keyword:

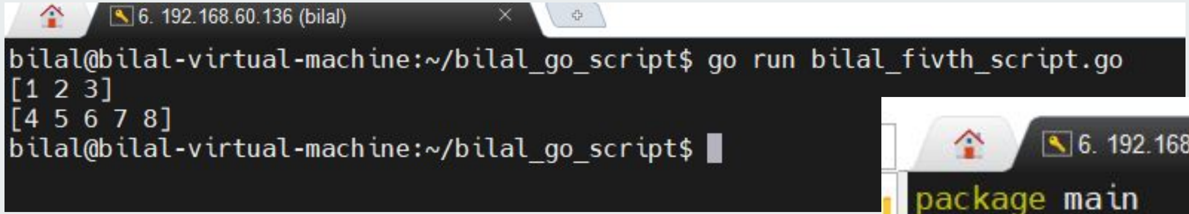


```
var array_name = [length]datatype{values} // here length is defined

or

var array_name = [...]datatype{values} // here length is inferred
```

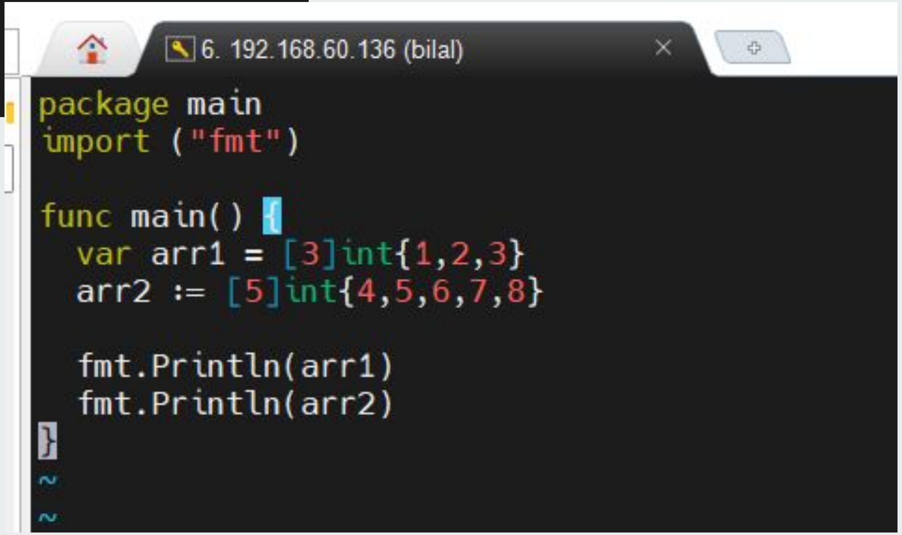


```
array_name := [length]datatype{values} // here length is defined

or

array_name := [...]datatype{values} // here length is inferred
```

Example 4



```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_fivth_script.go
[1 2 3]
[4 5 6 7 8]
bilal@bilal-virtual-machine:~/bilal_go_script$
```



```
package main
import ("fmt")

func main() {
  var arr1 = [3]int{1,2,3}
  arr2 := [5]int{4,5,6,7,8}

  fmt.Println(arr1)
  fmt.Println(arr2)
}
~
~
```

Go Conditions

→ Conditional statements are used to perform different actions based on different conditions.

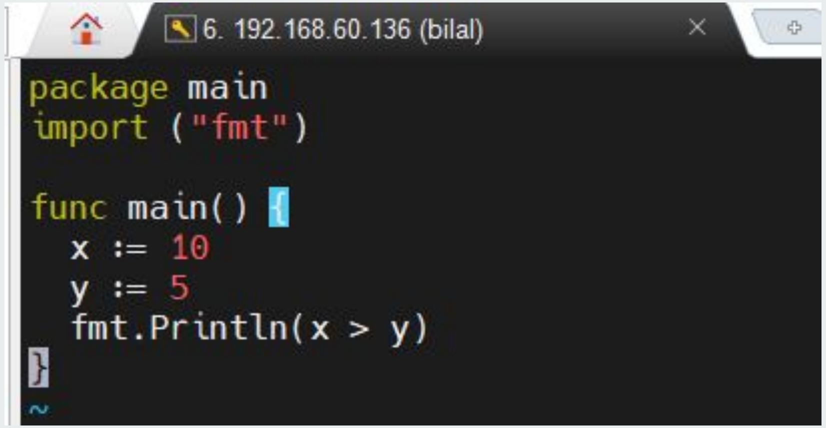
Go supports the usual comparison operators from mathematics:

- Less than **<**
- Less than or equal **<=**
- Greater than >
- Greater than or equal **>=**
- Equal to **==**
- Not equal to **!=**

Additionally, Go supports the usual logical operators:

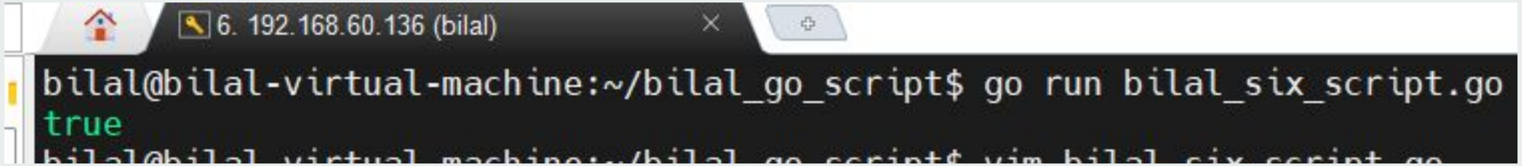
- Logical **AND &&**
- Logical **OR ||**
- Logical **NOT !**

Example 5


6. 192.168.60.136 (bilal)

```
package main
import ("fmt")

func main() {
  x := 10
  y := 5
  fmt.Println(x > y)
}
~
```

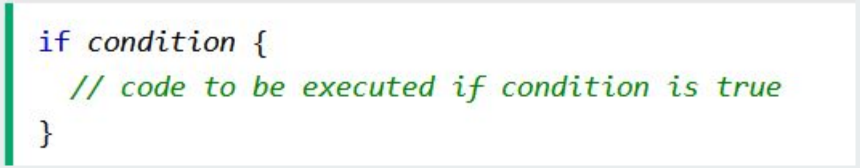
→ fmt.Println(x != y)
→ fmt.Println((x > y) && (y > z))
→ fmt.Println((x == y) || z)


6. 192.168.60.136 (bilal)

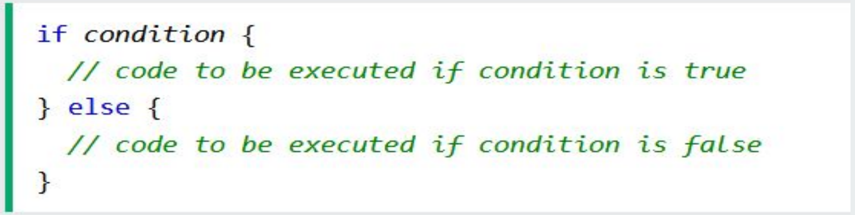
```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_six_script.go
true
```

if Statement

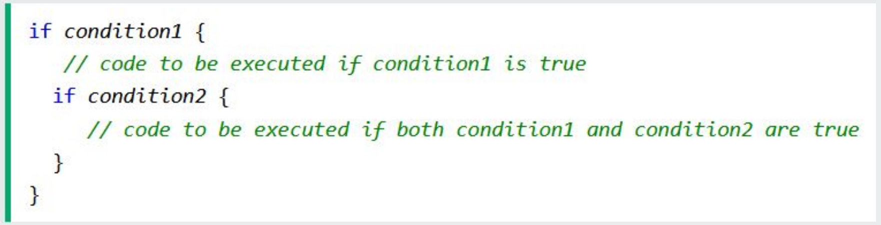
→ Use the if statement to specify a block of Go code to be executed if a condition is true.



```
if condition {
  // code to be executed if condition is true
}
```

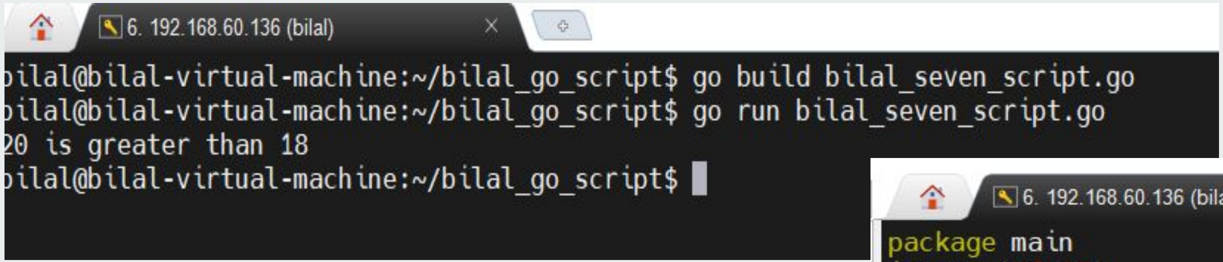


```
if condition {
  // code to be executed if condition is true
} else {
  // code to be executed if condition is false
}
```



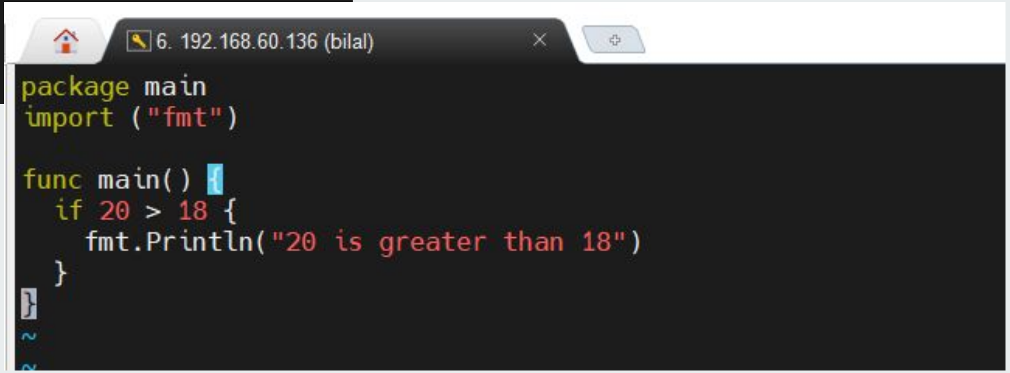
```
if condition1 {
   // code to be executed if condition1 is true
  if condition2 {
     // code to be executed if both condition1 and condition2 are true
  }
}
```

Example 6



6. 192.168.60.136 (bilal)

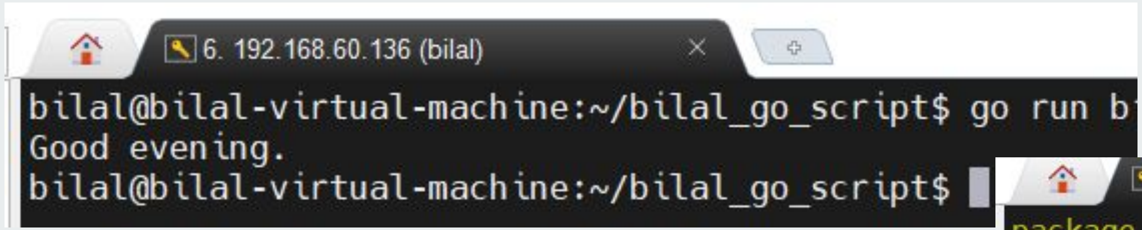
```
ilal@bilal-virtual-machine:~/bilal_go_script$ go build bilal_seven_script.go
ilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_seven_script.go
0 is greater than 18
ilal@bilal-virtual-machine:~/bilal_go_script$
```



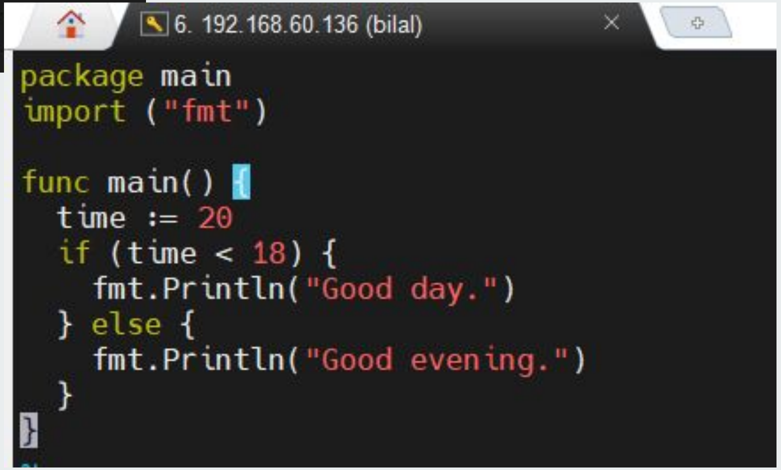
6. 192.168.60.136 (bilal)

```
package main
import ("fmt")

func main() {
  if 20 > 18 {
    fmt.Println("20 is greater than 18")
  }
}
~
~
```

Example 7

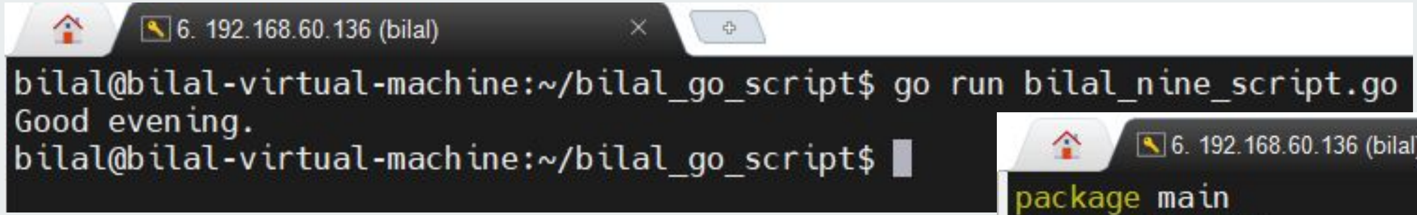


```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run b
Good evening.
bilal@bilal-virtual-machine:~/bilal_go_script$
```



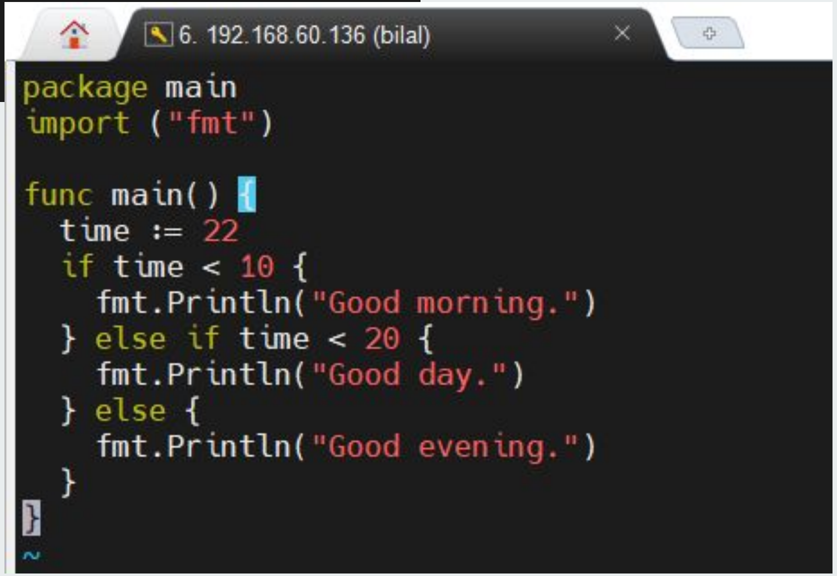
```
package main
import ("fmt")

func main() {
  time := 20
  if (time < 18) {
    fmt.Println("Good day.")
  } else {
    fmt.Println("Good evening.")
  }
}
```

Example 8



6. 192.168.60.136 (bilal)

```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_nine_script.go
Good evening.
bilal@bilal-virtual-machine:~/bilal_go_script$ 
```



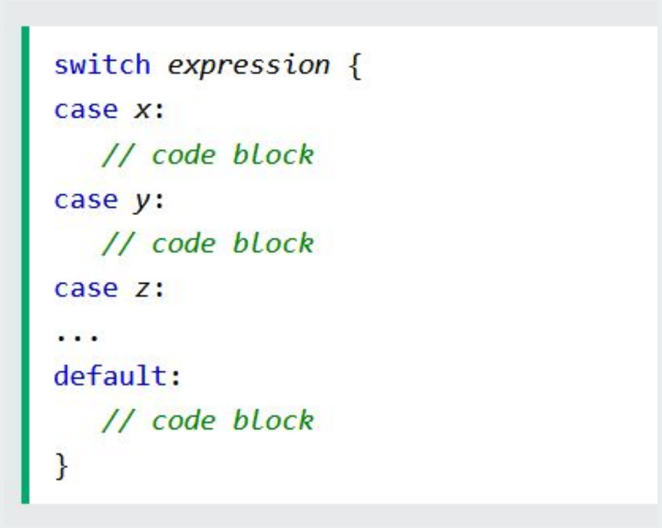
6. 192.168.60.136 (bilal)

```
package main
import ("fmt")

func main() {
  time := 22
  if time < 10 {
    fmt.Println("Good morning.")
  } else if time < 20 {
    fmt.Println("Good day.")
  } else {
    fmt.Println("Good evening.")
  }
}
~
```

switch Statement

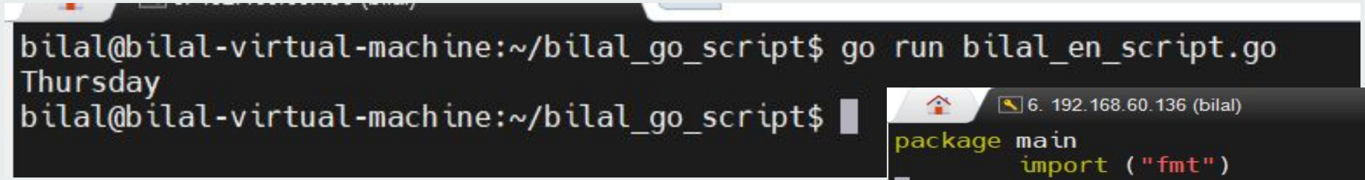
→ Use the switch statement to select one of many code blocks to be executed.

The switch statement in Go is similar to the ones in C, C++, Java, JavaScript, and PHP. The difference is that it only runs the matched case so it does not need a break statement.

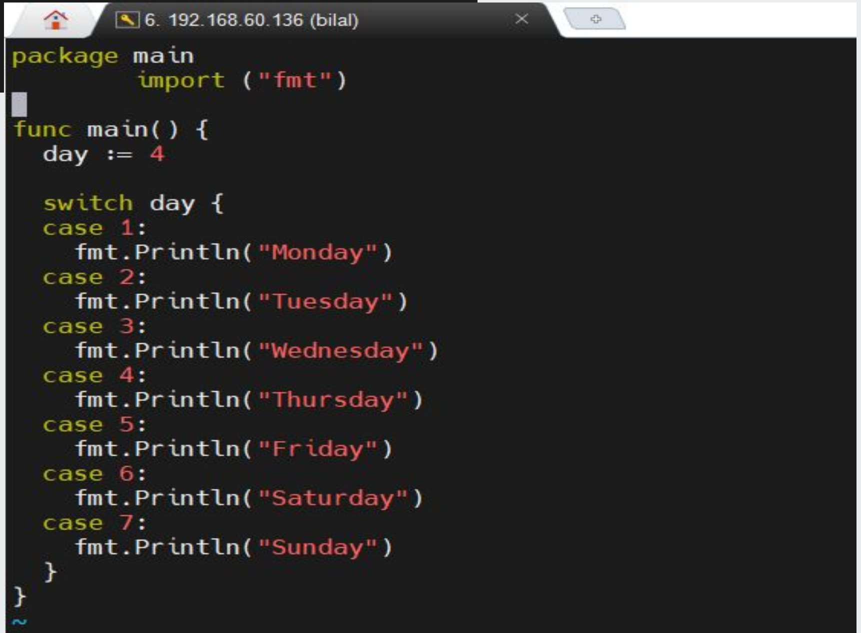


```
switch expression {
case x:
   // code block
case y:
   // code block
case z:
...
default:
   // code block
}
```

Example 9



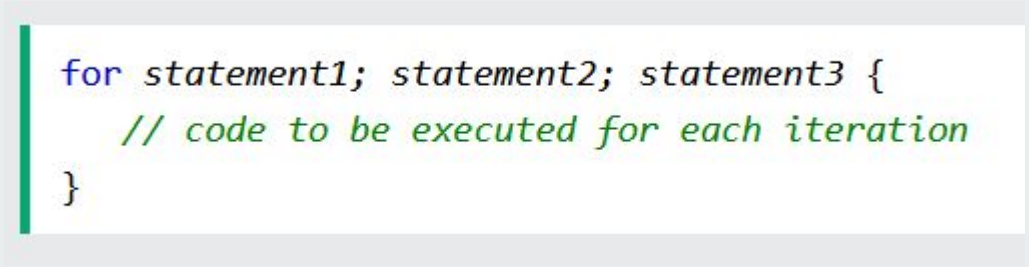
```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_en_script.go
Thursday
bilal@bilal-virtual-machine:~/bilal_go_script$
```



```
package main
        import ("fmt")

func main() {
  day := 4

  switch day {
  case 1:
    fmt.Println("Monday")
  case 2:
    fmt.Println("Tuesday")
  case 3:
    fmt.Println("Wednesday")
  case 4:
    fmt.Println("Thursday")
  case 5:
    fmt.Println("Friday")
  case 6:
    fmt.Println("Saturday")
  case 7:
    fmt.Println("Sunday")
  }
}
~
```

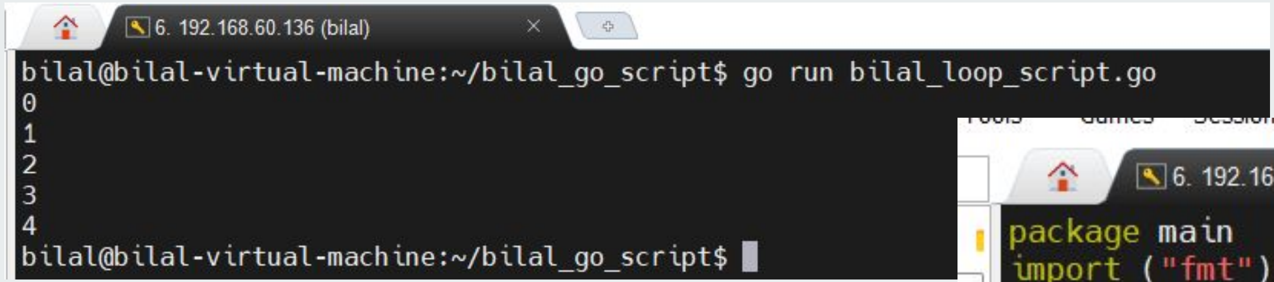
Loop

- → The for loop loops through a block of code a specified number of times
- → The for loop is the only loop available in Go.



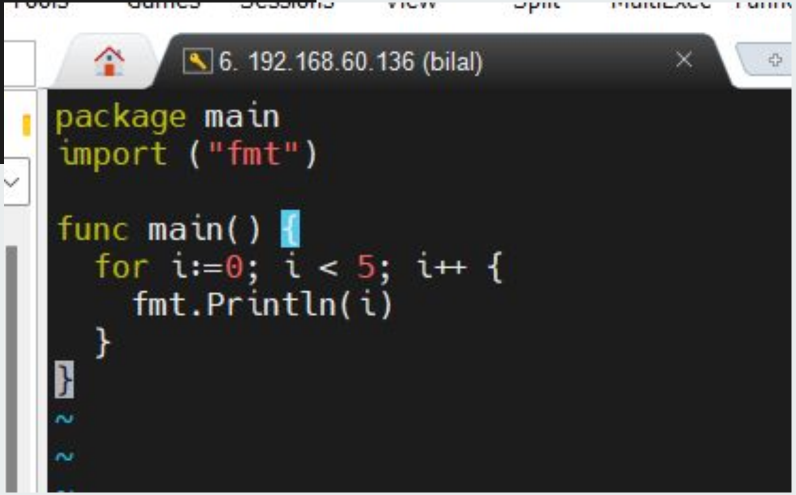
```
for statement1; statement2; statement3 {
   // code to be executed for each iteration
}
```

Example 10



6. 192.168.60.136 (bilal)

```
bilal@bilal-virtual-machine:~/bilal_go_script$ go run bilal_loop_script.go
0
1
2
3
4
bilal@bilal-virtual-machine:~/bilal_go_script$ 
```

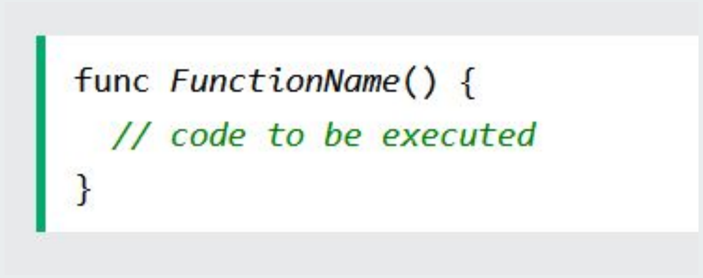


6. 192.168.60.136 (bilal)

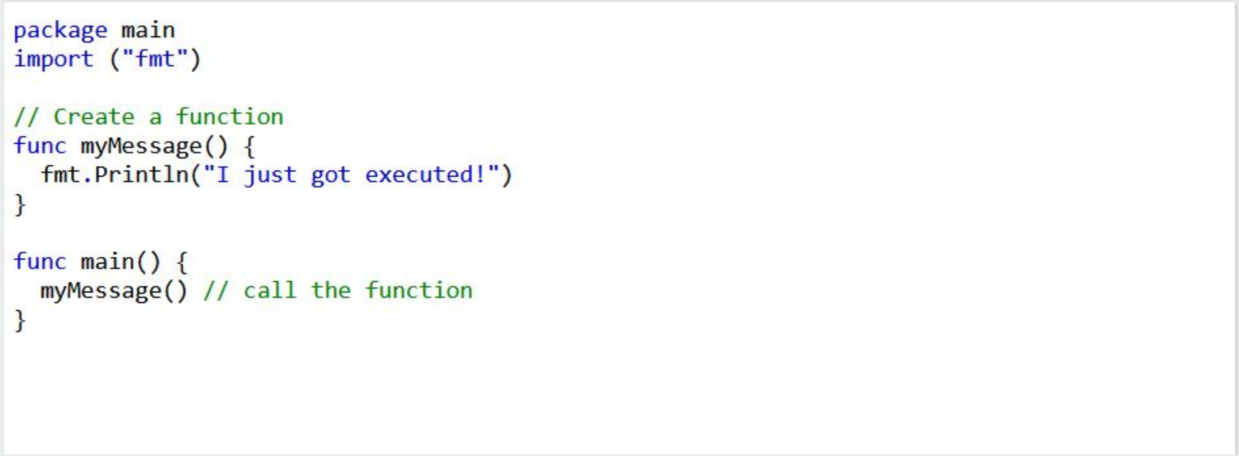
```
package main
import ("fmt")

func main() {
  for i:=0; i < 5; i++ {
    fmt.Println(i)
  }
}
~
~
```

Go Functions

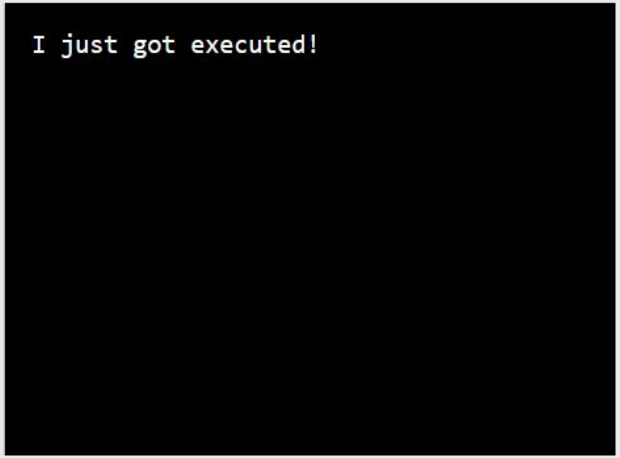
- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.



```go
func FunctionName() {
  // code to be executed
}
```



```go
package main
import ("fmt")

// Create a function
func myMessage() {
  fmt.Println("I just got executed!")
}

func main() {
  myMessage() // call the function
}
```



```
I just got executed!
```