Golang Tutorial – Learn Golang by Examples

Last updated on May 22,2019 *7.2K Views*

Paul
Research Analyst at edureka with a proficiency in Ethereum, Cybersecurity and Cryptography!

Golang has become one of the most trending languages in the developer community because go provides the best of both worlds, by striking a balance between dynamic and statically compiled languages. The code is easy to read, the specification is short, but even so, it includes a built-in web server! Throughout this Golang tutorial, not only will we be covering the reasons that make Go such a fantastic language, but also go over a majority of its concepts.

Following are the topics that will be covered throughout this Golang Tutorial:

Why Learn Golang?

Before you learn any language, you should always know why you're learning the language. Every language serves a special purpose and the very same goes for Golang. So why exactly should someone learn Golang? Well, Golang was built by **Google** to help solve problems faced by developers at google! At Google, lots of big programmes are built for big server software that runs on massive clusters. Before Go, Google was using languages like **C++** and **Java** to solve problems, but these languages didn't really have the fluidity and ease of construction that was needed for problems of such scale. Keeping these things in mind, a few years ago **Ken Thompson** and **Robert Greaser** thought about building a new language that would be good for these kinds of problems, and hence Golang was born. Basically, if you're trying to solve a problem of enormous scale or just need efficient and scalable code, ***Go should be your Go to language!***

What makes Golang special?

The existing languages have been good enough for large scale programming until recent years, but there have been big changes in the computing environment in the last decade which has included a lot of networking, a lot of cluster computing or the cloud as common folk might know it by. The languages that were being used to implement services until now are at least 10-20 old, so there are a lot of properties of a modern computing environment that these languages don't address directly. Therefore having a modern language that works really well in the modern computing environment is actually important, but you also want it to be efficient because you're going to be running these on hundreds or maybe even thousands of machines. You also don't want to waste resources by having an inefficient interpreter or some of the problems that generally come up with virtual machine implementations.

GoLang ticks all these boxes and hence has garnered the much-deserved fame in the developer's community.

Hello World: First Golang Programme

```
package main

import "fmt"

func main () {

fmt.Println("Hello World! This is my first Golang programme")

}
```

Let's go over the code line by line.

The first line is a package declaration. Every go programme must belong to a package and this particular programme belongs to the "main" package. Next, we import the "fmt" or format library which provides the developer with a wide array of functions that allows formatting of output. Then we create the main function which is the first function that is invoked when a go programme is executed. In the main function, we simply invoke the 'Println' function to print our statement.

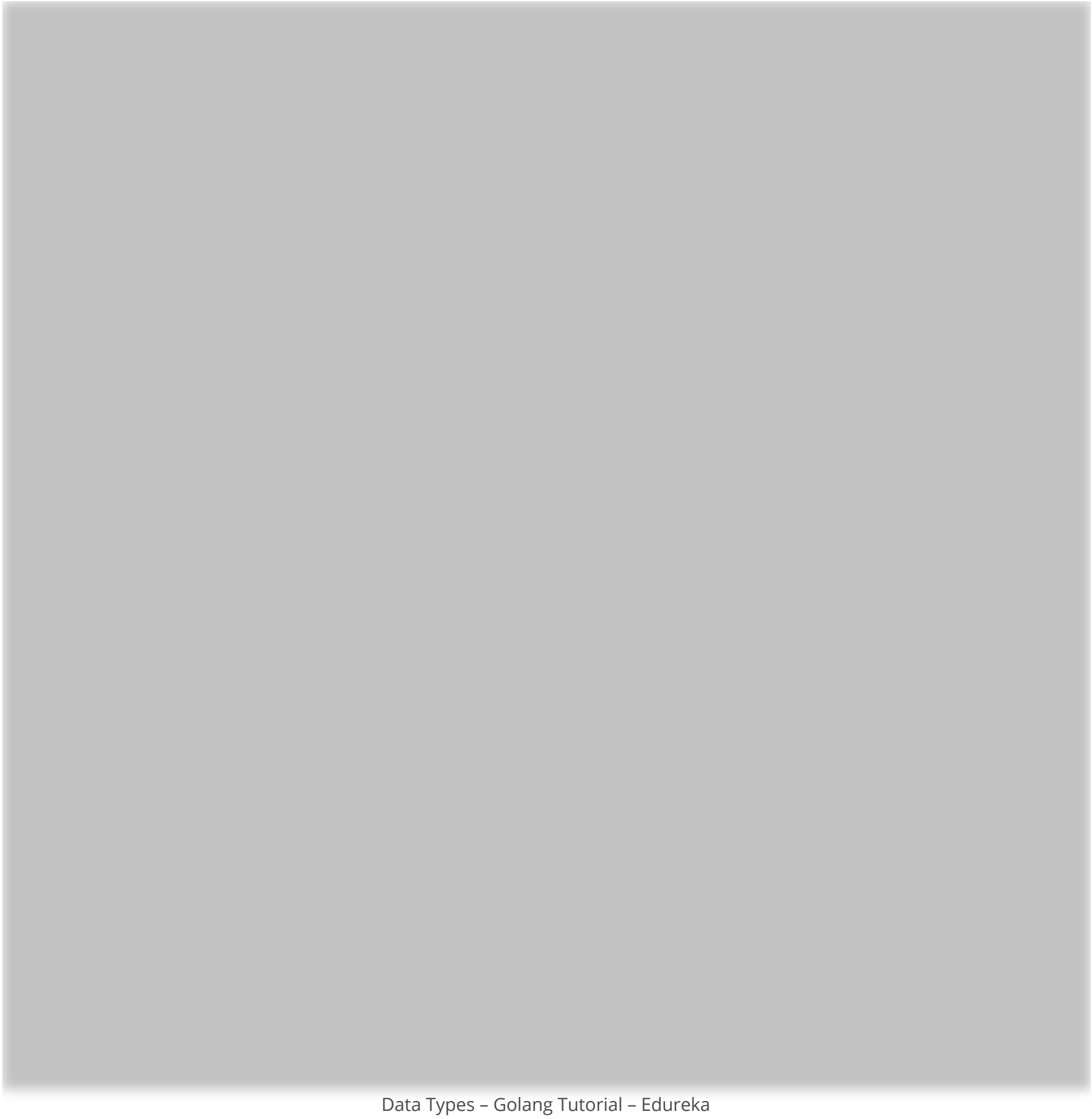
Variables and Constants

What exactly is a variable? A variable is nothing but a name given to a storage area that the programs can manipulate. Each variable in Go has a specific type, which determines the size and layout of the variable's memory, the range of values that can be stored within that memory, and the set of operations that can be applied to the variable. Next, **Constants** refer to fixed values that the program may not alter during its execution. These fixed values are also called literals. Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well. Constants are treated just like regular variables except that their values cannot be modified after their definition.

```
package main

import "fmt"

func main () {

var x int = 5 //declaration of variable x

var (
a = 10
b = 15 //declaration of multiple variables
)

y := 7 // shorthand declaration of variables

const pi float64 = 3.14272345 //declaration of constants

var name string = "Aryya Paul" //declaration of a string
}
```

Data Types

Golang has the three major types of operators that are prevalent in every programming language i.e arithmetic, relational and logical operators.



Data Types – Golang Tutorial – Edureka

Data Type	Range
uint8	0 to 255
uint16	0 to 65535
uint32	0 to 4294967295
uint64	0 to 18446744073709551615

Data Type	Range
int8	-128 to 127
int16	-32768 to 32767
int32	-2147483648 to 2147483647
int64	-9223372036854775808 to 9223372036854775808

Operators

Golang has the three general types of operators that are prevalent in every major programming language.

Operators – Golang Tutorial – Edureka

Pointers

Now it's time to see how pointers work in Go. Pointers in Go are easy and fun to learn. Some Go programming tasks are performed more easily with pointers, and other tasks, such as call by reference, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect Go programmer. As you know, every variable is a memory location and every memory location has its address defined which can be accessed using an ampersand (&), which denotes an address in memory.

```
package main

import "fmt"
// POINTERS

func main() {

// We pass the value of a variable to the function
x := 0
changeVal(x)
fmt.Println("x =",x)

// If we pass a reference to the variable we can change the value
// in a function
changeVal(&x)
fmt.Println("x =",x)

// Get the address x points to with &
fmt.Println("Memory Address for x =", &x)

// We can also generate a pointer with new

}

func changeVal(x int) {

// Has no effect on the value of x in main()
x = 2

}

// * signals that we are being sent a reference to the value
func changeXValNow(x *int){

// Change the value at the memory address referenced by the pointer
// * gives us access to the value the pointer points at

*x = 2 // Store 2 in the memory address x refers to

}

}
```

Printf

Printf is used for format printing our outputs in Golang. It is a part of the format (fmt) library. Below is a table listing out the noteworthy uses of Printf function.

Function	Usage
fmt.Printf(“%f “,pi)	%f is for floats
fmt.Printf(“%.3f “,pi)	You can also define the decimal precision of a float
fmt.Printf(“%T “,pi)	%T prints the data type
fmt.Printf(“%t “,isOver40)	%t prints booleans
fmt.Printf(“%d “,100)	%d is used for integers
fmt.Printf(“%b “,100)	%b prints (100) in binary
fmt.Printf(“%c “,44)	%c prints the character associated with a keycode
fmt.Printf(“%x “,17)	%x prints in hexcode
fmt.Printf(“%e “,pi)	%e prints in scientific notation

Ok, it’s time we move on to loops.

Loops

If you’re new to programming, a loop is a basic iterating mechanism in computer science and it’s used mostly when you need to perform a repetitive pattern in programming. Now in most programming languages, there are three types of loops, namely for, while(exit controlled) and do-while(entry controlled) but Golang has only one type of loop that is the ‘for’ loop. The syntax of go allows while loops to be implemented with the syntax of a ‘for’ loop.

```
package main

import "fmt"

func main() {

// For loops

i := 1

for i <= 10 {
fmt.Println(i)

// Shorthand for i = i + 1

i++
}

// Relational Operators include ==, !=, <, >, <=, and >=

// You can also define a for loop like this, but you need semicolons

for j := 0; j < 5; j++ {

fmt.Println(j);

}

}
```

Decision Making

Decision making is a critical part of programming. In Golang, we can implement decision making using the 'if-else' and 'switch' keywords. Let's see how Golang implements decision making with this piece of code below:

```
package main

import "fmt"

func main() {

// If Statement

age := 15

if age >= 16 {
fmt.Println("Adult")
} else {
fmt.Println("Not an adult")
}

// You can use else if perform different actions, but once a match
// is reached the rest of the conditions aren't checked

if age >= 16 {
fmt.Println("in school")
} else if age >= 18 {
fmt.Println("in college")
} else {
fmt.Println("probably dead")
}

// Switch statements are used when you have limited options

switch age {
case 16: fmt.Println("Go Drive")
case 18: fmt.Println("Go Vote")
default: fmt.Println("Go Have Fun")
}

}
```

Arrays

An array is a data structure in programming that is used to containerize data of the same type. For example, if you were to store all the student name of a certain class, you would use a string array to store them all. The code below shows how arrays are implemented in Golang.

```
package main

import "fmt"

func main() {

// An Array holds a fixed number of values of the same type

var favNums2[5] float64

favNums2[0] = 163
favNums2[1] = 78557
favNums2[2] = 691
favNums2[3] = 3.141
favNums2[4] = 1.618

// You access the value by supplying the index number

fmt.Println(favNums2[3])

// Another way of initializing an array

favNums3 := [5]float64 { 1, 2, 3, 4, 5 }

// How to iterate through an array (Use _ if a value isn't used)

for i, value := range favNums3 {

fmt.Println(value, i)

}

// Slices are like arrays but you leave out the size

numSlice := []int {5,4,3,2,1}

// You can create a slice by defining the first index value to
// take through the last

numSlice2 := numSlice[3:5] // numSlice3 == [2,1]

fmt.Println("numSlice2[0] =", numSlice2[0])

// If you don't supply the first index it defaults to 0
// If you don't supply the last index it defaults to max

fmt.Println("numSlice[:2] =", numSlice[:2])

fmt.Println("numSlice[2:] =", numSlice[2:])

// You can also create an empty slice and define the data type,
// length (receive value of 0), capacity (max size)

numSlice3 := make([]int, 5, 10)

// You can copy a slice to another

copy(numSlice3, numSlice)

fmt.Println(numSlice3[0])

// Append values to the end of a slice

numSlice3 = append(numSlice3, 0, -1)

fmt.Println(numSlice3[6])

}
```

Maps

Besides arrays, we also have another data structure called “Maps” which maps unique keys to values. A key is an object that you use to retrieve a value at a later date. Given a key and a value, you can store the value in a Map object. After the value is stored, you can retrieve it by using its key.

```
package main

import "fmt"

func main() {

// A Map is a collection of key value pairs
// Created with varName := make(map[keyType] valueType)

presAge := make(map[string] int)

presAge["Narendra Modi"] = 42

fmt.Println(presAge["Narendra Modi"])

// Get the number of items in the Map

fmt.Println(len(presAge))

// The size changes when a new item is added

presAge["Rahul Gandhi"] = 43
fmt.Println(len(presAge))

// We can delete by passing the key to delete

delete(presAge, "Rahul Gandhi")
fmt.Println(len(presAge))

}
```

Next up, let's move on to functions.

Functions

A function is a group of statements that together perform a task. Every Go program has at least one function, which is main(). You can divide your code into separate functions. How you divide your code among different functions is up to you, but logically, the division should be such that each function performs a specific task. A function declaration tells the compiler about a function name, return type, and parameters.

```
package main

import "fmt"

func main () {

fmt.Println("5 + 4 = ", add(5,4))
fmt.Println(subtract(1,2,3,4,5))

}

func add(a,b int) int {

return a+b

}

func subtract(args ... int) {

sub := 0

for _, value := range args {
sub -= value
}

return sub

}
```

Recursion

Recursion is the process of repeating items in a self-similar way. The same concept applies to programming languages as well. If a program allows calling a function inside the same function, then it is called a 'recursive function' call. GoLang supports recursion i.e, it allows a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise, it will go on to become an infinite loop.

```
package main

import "fmt"

func main () {

fmt.Println(factorial(5))

}

func factorial(num int) int {

if num == 0 {

return 1
}
return num * factorial(num-1)
}
```

Defer, Panic & Recover

Defer statement defers the execution of a function until the surrounding function returns. They are generally used to execute necessary closing statements, for example closing a file after you are done with it. Multiple defers are pushed into stack and executes in Last In First Out (LIFO) order. Defer generally used to clean up resources like a file, database connection etc.

Panic is similar to throwing an exception like other languages. Generally when a panic is called, then the normal execution flow stops immediately, but the deferred functions are executed normally. It is a built-in function in Golang.

Recover is another built-in function in go. It helps to regain the normal flow of execution after a panic. Generally, it used with a defer statement to recover panic in a goroutine.

```
package main

import "fmt"

func main() {

// Defer executes a function after the inclosing function finishes
// Defer can be used to keep functions together in a logical way
// but at the same time execute one last as a clean up operation
// Ex. Defer closing a file after we open it and perform operations

defer printTwo()
printOne()

// Use recover() to catch a division by 0 error

fmt.Println(Div(3, 0))
fmt.Println(Div(3, 2))

// We can catch our own errors and recover with panic & recover

demPanic()

}


func factorial(num int) int {
if num == 0 {
return 1
}
return num * factorial(num - 1)
}

// Used to demonstrate defer

func printOne(){ fmt.Println(1)}

func printTwo(){ fmt.Println(2)}

// If an error occurs we can catch the error with recover and allow
// code to continue to execute

func Div(num1, num2 int) int {
defer func() {
fmt.Println(recover())
}()
solution := num1 / num2
return solution
}

// Demonstrate how to call panic and handle it with recover

func demPanic(){

defer func() {

// If I didn't print the message nothing would show

fmt.Println(recover())

}()
panic("PANIC")

}
```

Structure

Go allow you to define variables that can hold several data items of the same kind. A structure is another user-defined data type available in Go programming, which allows you to combine data items of different kinds. Structures are used to represent a record. Suppose you want to keep track of the books in a library. You might want to track the following attributes of each book −

- Title
- Author
- Subject
- Book ID

In such a scenario, structures are highly useful. To define a structure, you must use type and struct statements. The struct statement defines a new data type, with multiple members for your program. The type statement binds a name with the type which is a struct in our case.

```
package main

import "fmt"

// STRUCTS

func main() {

rect1 := Rectangle{height: 10, width: 10}

fmt.Println("Rectangle is", rect1.width, "wide")

fmt.Println("Area of the rectangle =", rect1.area())

}

type Rectangle struct{

height float64
width float64
}

func (rect *Rectangle) area() float64{

return rect.width * rect.height

}
```

Interface

Go programming provides another data type called interfaces which represents a set of method signatures. The struct data type implements these interfaces to have method definitions for the method signature of the interfaces.

```
package main

import "fmt"
import "math"

// STRUCTS AND INTERFACES

func main() {

rect := Rectangle{20, 50}
circ := Circle{4}

fmt.Println("Rectangle Area =", getArea(rect))
fmt.Println("Circle Area =", getArea(circ))

}

// An interface defines a list of methods that a type must implement
// If that type implements those methods the proper method is executed
// even if the original is referred to with the interface name

type Shape interface {
area() float64
}

type Rectangle struct{
height float64
width float64
}

type Circle struct{
radius float64
}

func (r Rectangle) area() float64 {
return r.height * r.width
}

func (c Circle) area() float64 {
return math.Pi * math.Pow(c.radius, 2)
}

func getArea(shape Shape) float64{

return shape.area()

}
```

Simple Webserver with Go

Go also provides us with the ability to set up a simple web server in a matter of seconds. It goes to show how robust and powerful Go libraries are.

```
package main

import (
"fmt"
"net/http"
)

// CREATE A HTTP SERVER

// http.ResponseWriter assembles the servers response and writes to
// the client
// http.Request is the clients request
func handler(w http.ResponseWriter, r *http.Request) {

// Writes to the client
fmt.Fprintf(w, "Hello World
")
}

func handler2(w http.ResponseWriter, r *http.Request) {
fmt.Fprintf(w, "Hello Earth
")
}

func main() {

// Calls for function handlers output to match the directory /
http.HandleFunc("/", handler)

// Calls for function handler2 output to match directory /earth
http.HandleFunc("/earth", handler2)

// Listen to port 8080 and handle requests
http.ListenAndServe(":8080", nil)
}
```

That's it for this Golang Tutorial blog. I hope you guys enjoyed reading it and leave confident enough to practice the fundamentals of Golang on your own. Do stay tuned for more Golang related blogs!

Got a question for us? Please mention it in the comments section of "Golang Tutorial" and we will get back to you.

ALSO ON **HTTPS://WWW.EDUREKA.CO/BLOG/**

Find-S Algorithm In Machine Learning: …
3 months ago · 1 comment
This article covers the concept of find-s algorithm in machine learning. It …

Social Media Marketing: An …
2 months ago · 3 comments
In the world of smartphones and Artificial Intelligence, it is quite important to …

What is On Page SEO? Ultimate Guide to …
2 months ago · 1 comment
What is On Page SEO? Know Its working in Detail The way you optimize …

5 Domains that solves the problem
4 months ago · 2 comments
DevOps has become the latest buzzword in the tech industry as of now. …

How to Use A AWS Commar
5 months ago · 1
AWS is one of th cloud providers a for 47.8% of the

3 Comments https://www.edureka.co/blog/ Disqus' Privacy Policy Login

Recommend 1 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

QSS Technosoft • 2 months ago
Golang is an open-source programming language designed by Google.Leading golang development company that Build highly interactive scalable apps at lightning-fast speed with golang developers. So, Go through them if you want to get reliable services.
• Reply • Share ›

varun • 5 months ago
What is the link to access video for GoLang
• Reply • Share ›

raj • 10 months ago
Golang in minutes..Excellant!!!!
• Reply • Share ›

Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data

Trending Courses

DevOps Certification Training
66k Enrolled Learners
Weekend/Weekday
Live Class
Reviews
5 (26300)

AWS Architect Certification Training
68k Enrolled Learners
Weekend/Weekday
Live Class
Reviews
5 (27050)

Python Certification Training for Data Scienc …
59k Enrolled Learners
Weekend/Weekday
Live Class
Reviews
5 (23600)

Cybersecurity Cer Course
15k Enrolled Learne
Weekend
Live Class
Reviews
5 (5850

Browse Categories

Artificial Intelligence | BI and Visualization | Big Data | Blockchain | Cloud Computing | Cyber Security | Data Science | Data Warehousing and ETL | Databases | DevOps | Digital Marketing | Front End Web Development | Mobile Development