***HOW TO GO!***

[***https://go.dev/doc/tutorial/getting-started***](https://go.dev/doc/tutorial/getting-started)

[***https://www.javatpoint.com/go-tutorial***](https://www.javatpoint.com/go-tutorial)

[***https://www.geeksforgeeks.org/golang/***](https://www.geeksforgeeks.org/golang/)

[***https://gobyexample.com/***](https://gobyexample.com/)

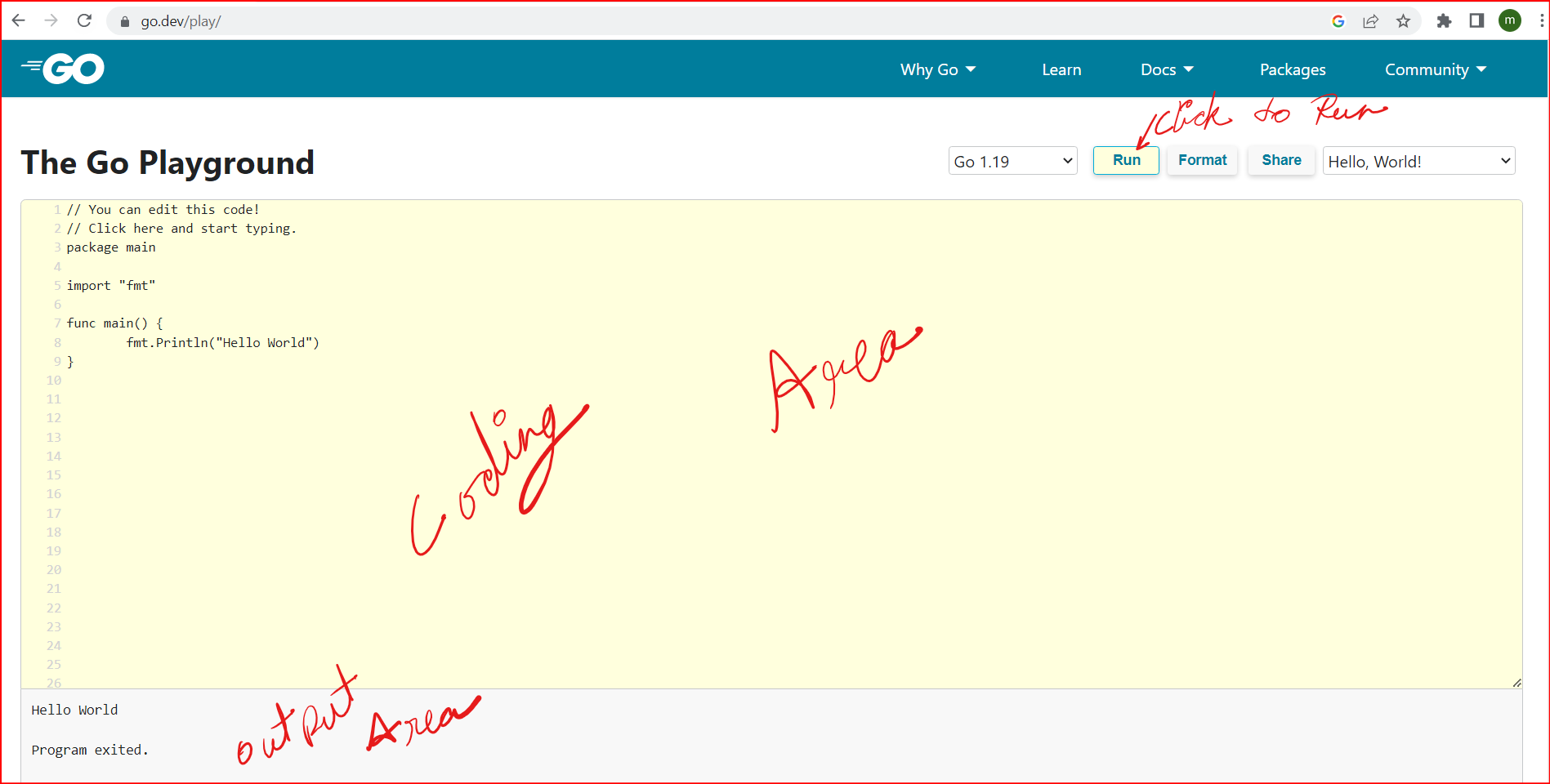
***Prerequisite***

\* Basic knowledge of programming.

\* Computer / Laptop.

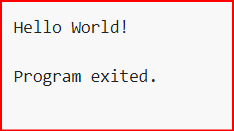
**GO play ground** -> <https://go.dev/play/>

Go playground is an online space where we can code and learn GO (GOLANG).



**A Sample Program** to “Hello World!”

Text

Description automatically generated

**Understanding the Syntax:**

1. First Line must be the package name.
2. Followed by imports.
3. And then rest of the code.

**What is package and how to write**: -

In Golang each piece of code belongs to some package.

The purpose of a package is to design and maintain a large number of programs by grouping related features together into single units so that they can be easy to maintain and understand and independent of the other package programs. ([link](https://www.geeksforgeeks.org/packages-in-golang/))

package <package\_name>

<package\_name> can be any thing of your choice

e.g.,

package main,

package constant

**What is import and how to write**: -

Import is used to make code in one package available in another.

import “<package\_name>”

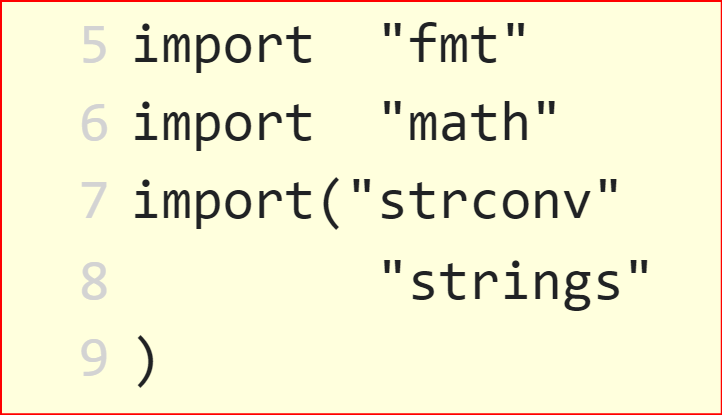
import “<package\_name>”

import ( “<package\_name>”

“<package\_name>”

…

)



**What is function and how to write: -**

A function is a group of statements that together perform a task.([link](https://www.tutorialspoint.com/go/go_functions.htm#:~:text=A%20function%20is%20a%20group,your%20code%20into%20separate%20functions.))

1.

func <function\_name>(<ipn> <ipt>){

}

2.

func <function\_name>(<ipn> <ipt>) <rpt> {

}

3.

func <function\_name>(<ipn> <ipt>, <ipn> <ipt>, <ipn> <ipt>) (<rpt>,<rpt>,<rpt>) {

}

4.

func <function\_name>(<ipn> <ipt>, <ipn> <ipt>) (<rpn> <rpt>, <rpn> <rpt>) {

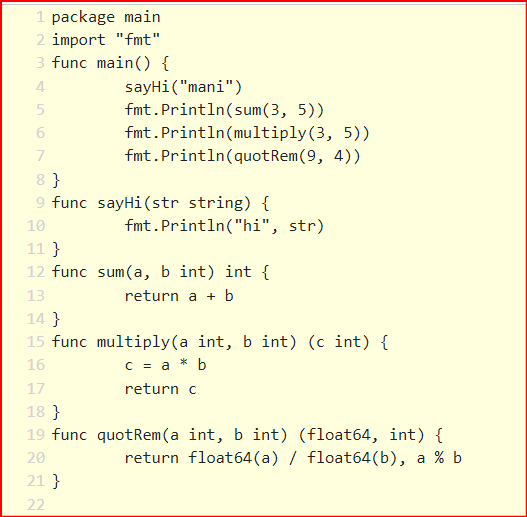
}

ipn => input\_parameter\_name

ipt => input \_parameter\_type

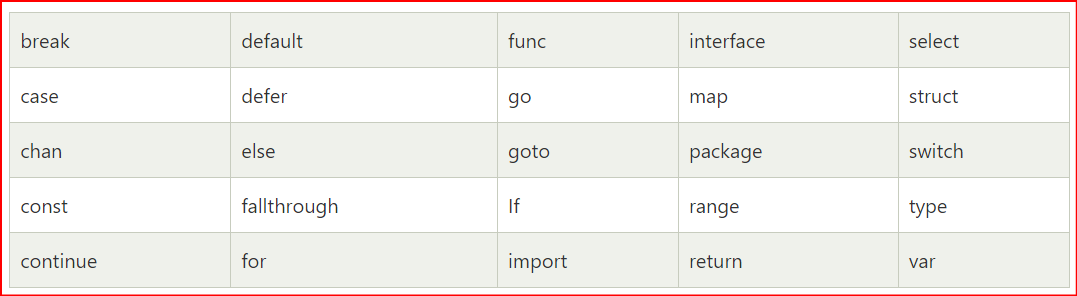
rpn => return\_parameter\_name

rpt => return\_parameter\_type

Text

Description automatically generated

**Keywords in Golang: -**



Table

Description automatically generated

**Variable declaration in Golang: -**

1.

var <variable\_name> <variable\_type>

var num1 int

var num1, num2 int

2.

var <variable\_name> <variable\_type> = <value>

var num1,num2 int = 4,5

3.

<variable\_name> := <value>

num1 := 4

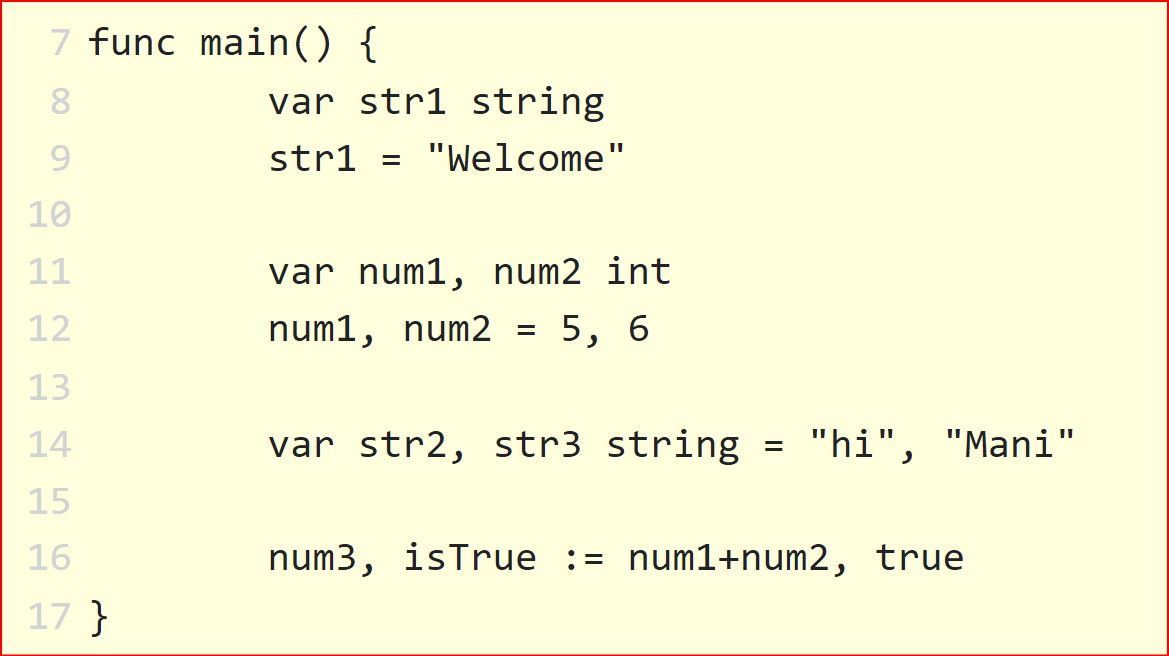
num1,num2 :=4,5

\* We can’t declare two different kind of variable in same line using example 1 & 2 but can do with example 3.

**Constants**

const <variable\_name> = <value>

const <variable\_name> <variable\_type> = <value>



**Controls**

**if-else**

1. *only if*

if <condition> {

}­­­­­

if <condition1> <logical\_operator> <condition2>{

}

\* Logical\_operator => &&, ||

1. *if else*

if <condition>{

} else{

}

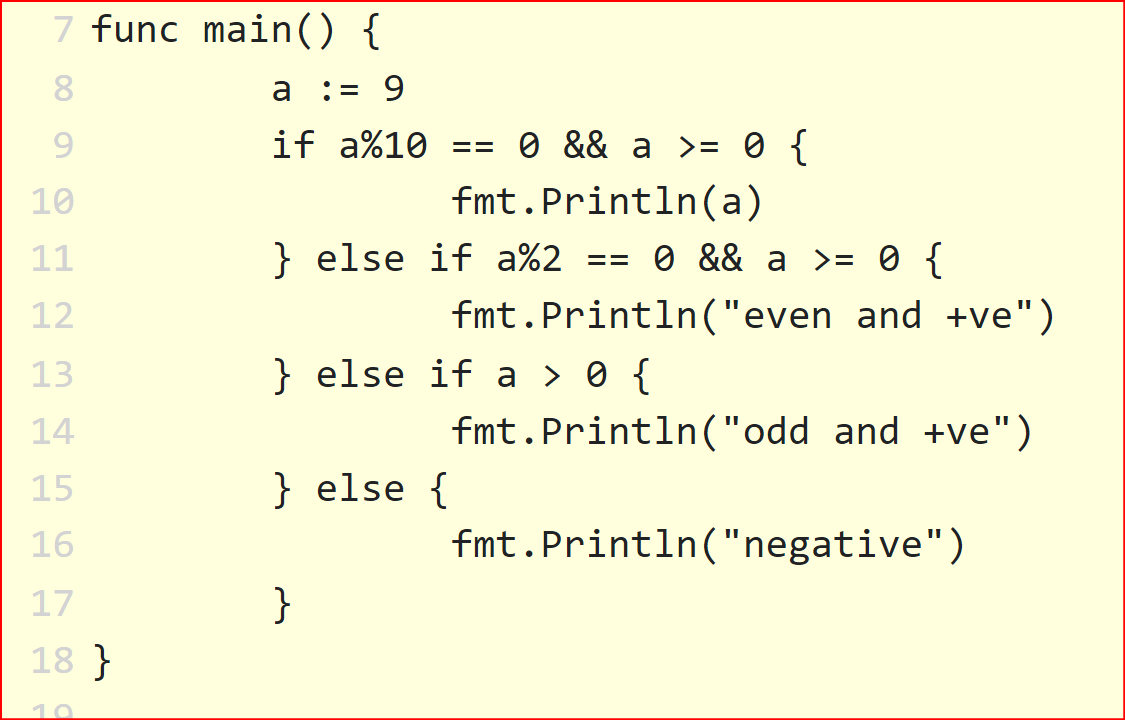
1. *if else - ladder/chain*

if <condition>{

} else if <condition>{

} else{

}



**Loop**

**for loop**

for <variable declaration/assignment> ; <condition>; <operation> {

}

for <condition> {

}

\* There is no while and do while loop in Golang (And which make us realize that you can do any thing with for loop and some conditions).

**for range**

var datas [5]int

for <index>, <value> := range data{

}

Text

Description automatically generated

**Shape

Description automatically generated**

**Continue/ Break**

Continue is used just similar to other language to skip code.

Break is used to break out of loop.

Text, letter

Description automatically generatedText

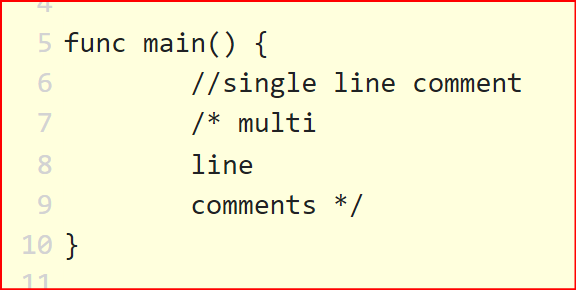
Description automatically generated

**Comments**

// single line comments

/\* multi line

Comments\*/

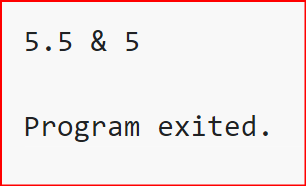
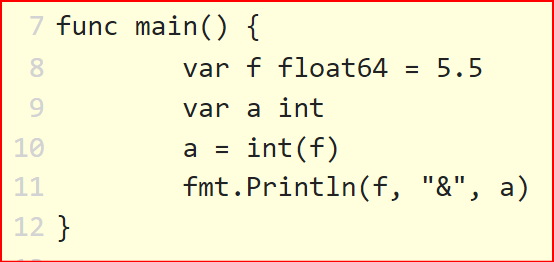


**Type Casting**

var <variable\_name> <target\_type>(<input\_variable>)

val := 5.5

v := int(val)



**Switch Case**

switch (<input>){

case <val1>:

fallthrough

case <val2>:

default:

Text

Description automatically generatedText

Description automatically generated

\* falltrhough, is used to go in next case, as in Golang case don’t need break.

**Closure**

closure is a function with similar property of a function which is declared inside another function and can be used within the scope.

Text

Description automatically generated Text

Description automatically generated

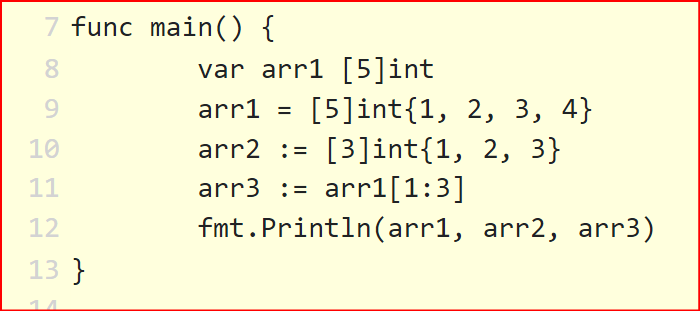
**Array**

var <variable\_name> [<size>]<type>

<variable\_name> := make([]<type> ,<size>)

**Slice**

Slice is dynamically sized array.



A screenshot of a computer

Description automatically generated with low confidence

**Variadic Functions**

Function with variable arguments of similar type.

It must be the last argument of a function.

A function must have only one type of variable argument

func <func\_name> ( <name> …<type>)(<type>){

}

Text, letter

Description automatically generated

Text

Description automatically generated

**Structure**

type <structure\_name> struct {

<name> <type>

}

**Embedded Structure**

Structure inside structure is called embedded structure.

A picture containing text

Description automatically generated

Text

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence

**Rune**

There is no char in Golang. Instead we have rune here.

var <variable\_name> rune = value

<variable\_name> := ‘value’

Text

Description automatically generated

A picture containing text

Description automatically generated

**Roroutine**

Goroutine is a light weighted tread of execution. Goroutines run synchronously with other threads.

To launch a goroutine we need to add “go” keyword before calling any function.

To completely execute your goroutine you need to explicitly mention your program to wait finish execution of routines.

go <function\_name>()

Text, letter

Description automatically generated

**Text

Description automatically generated**

**Channels**

Channels are the pipes that connect concurrent goroutines. You can send values into channels from one goroutine and receive those values into another goroutine. ([link](https://gobyexample.com/channels))

**Unbuffered channel**

Default channel is unbuffered, which means that a buffer will only accept data if there is a receiver.

<variable\_name> := make(chan <variable\_type>)

**Buffered channel**

Buffered channel can receive message up to its size without receiver at the other end.

<variable\_name> := make(chan <variabel\_type>, <channel\_size>)

Text, letter

Description automatically generated

**Select**

Just like switch case, select let you wait on multiple channel operation.

Text, letter

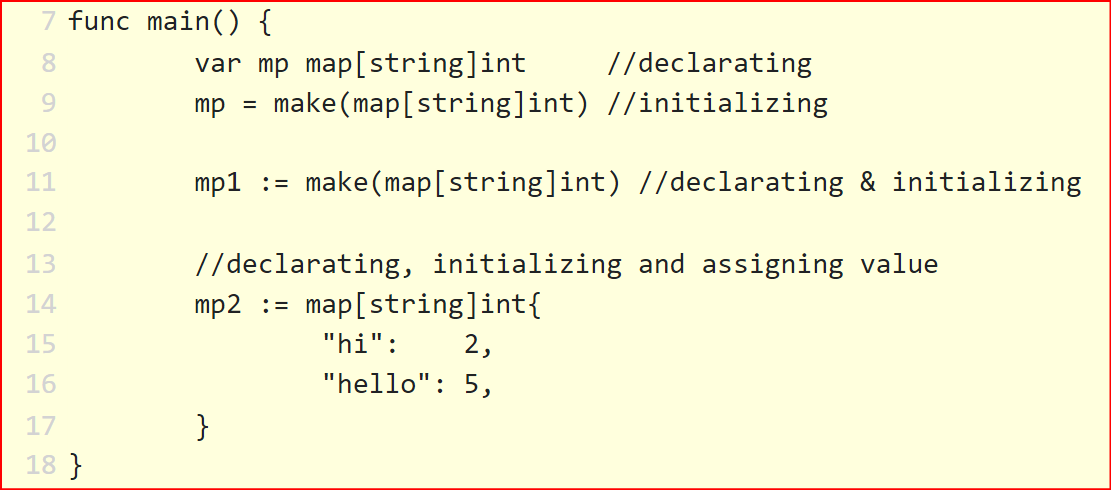
Description automatically generated

**Map**

Map is an unordered collection of key and its value

var <variable\_name> map[<key\_type>]<value\_type>

<variable\_name> := make(map[<key\_type>]<value\_type>)

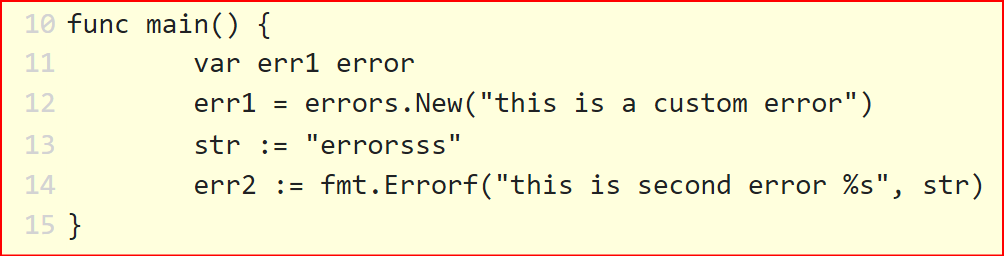


\* map need declaration as well as initialization

**Error**

In Golang we don’t have try/catch. To handle exceptions we use errors.

In case of program crash we have different mechanism called **differ**, **panic** and **recover.**



**Recover**

Recover is used to avoid unwanted termination of program caused due to error or panic.

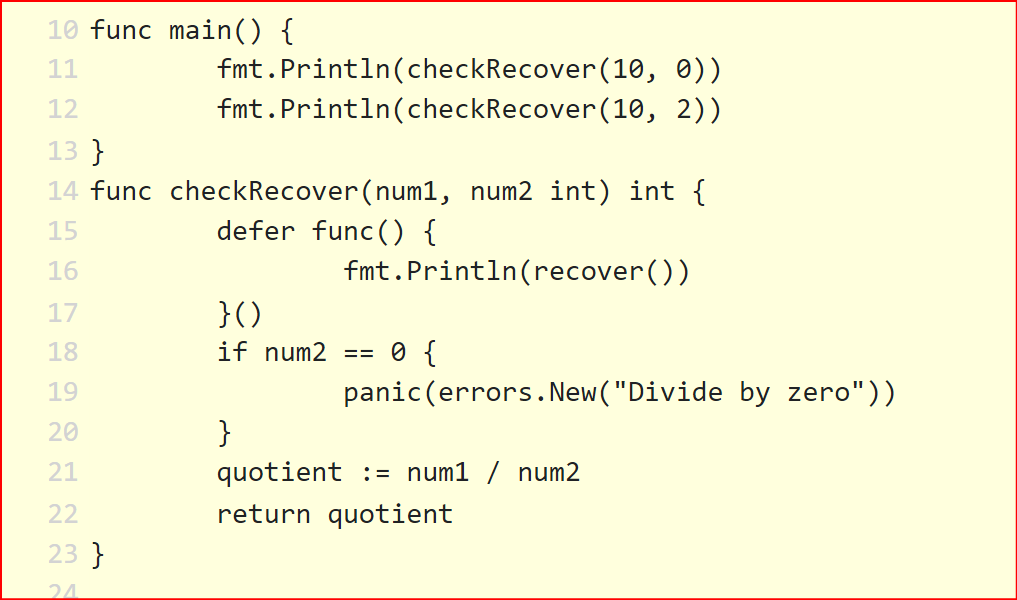
**Panic**

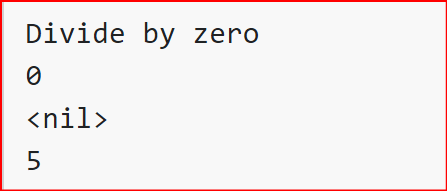
Panic is used to abort/terminate execution of program. Panic is used to handle error situation.

**Defer**

Defer keyword is used to postpone the execution of function or statement until the end of the function.

Defer can be used for cleaning purpose like close the opened files.





**Contacts**

**Name: -** *Manindra Narayan Singh*

**Email Id: -** [*mns.manindra@gmail.com*](mailto:mns.manindra@gmail.com)

**LinkedIn: -** [*https://www.linkedin.com/in/mani...*](https://www.linkedin.com/in/mani...)

**Mobile No: -** *+91 8299661294 (WhatsApp)*