HOW TO GO!

https://go.dev/doc/tutorial/getting-started

https://www.javatpoint.com/go-tutorial

https://www.geeksforgeeks.org/golang/

https://gobyexample.com/

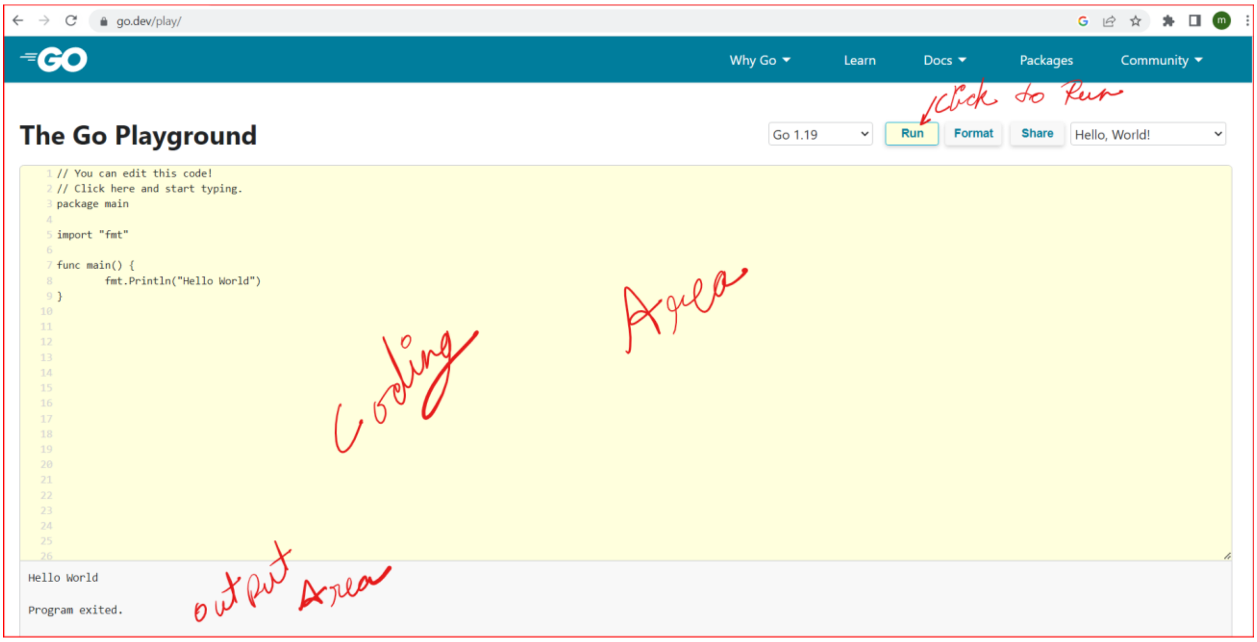
Prerequisite

* Basic knowledge of programming.

* Computer / Laptop.

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

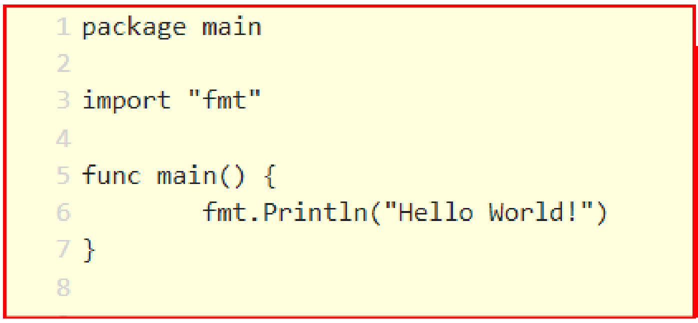
Go playground is an online space where we can code and learn GO (GOLANG).



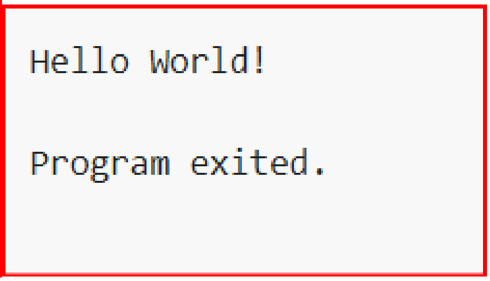
```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

func main() {
	fmt.Println("Hello World")
}
```

```
Hello World

Program exited.
```

A Sample Program to “Hello World!”



```
package main

import "fmt"

func main() {
	fmt.Println("Hello World!")
}
```



```
Hello World!

Program exited.
```

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.

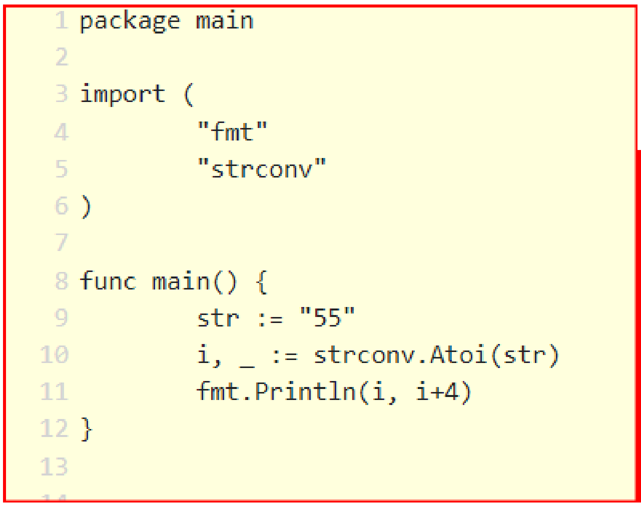
The purpose of a package is to design and maintain a large number of programs by grouping related features together into single units so that they can be easy to maintain and understand and independent of the other package programs. (link)

```
package <package_name>

<package_name> can be any thing of your choice
e.g.,
package main,
package constant
```

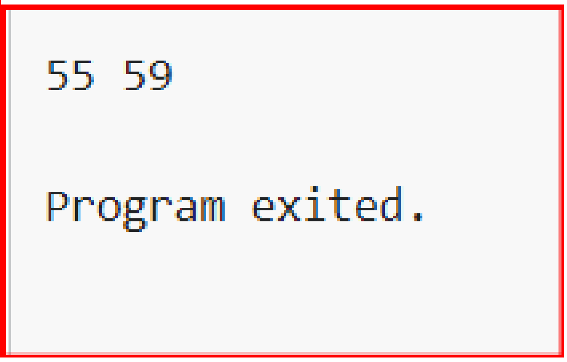
What is import and how to write: -

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

```
import “<package_name>”
import “<package_name>”

import ( “<package_name>”
         “<package_name>”
         …
         )
```



```go
package main

import (
        "fmt"
        "strconv"
)

func main() {
        str := "55"
        i, _ := strconv.Atoi(str)
        fmt.Println(i, i+4)
}
```



```
55 59

Program exited.
```

What is function and how to write: -

A function is a group of statements that together perform a task.(link)

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>) {

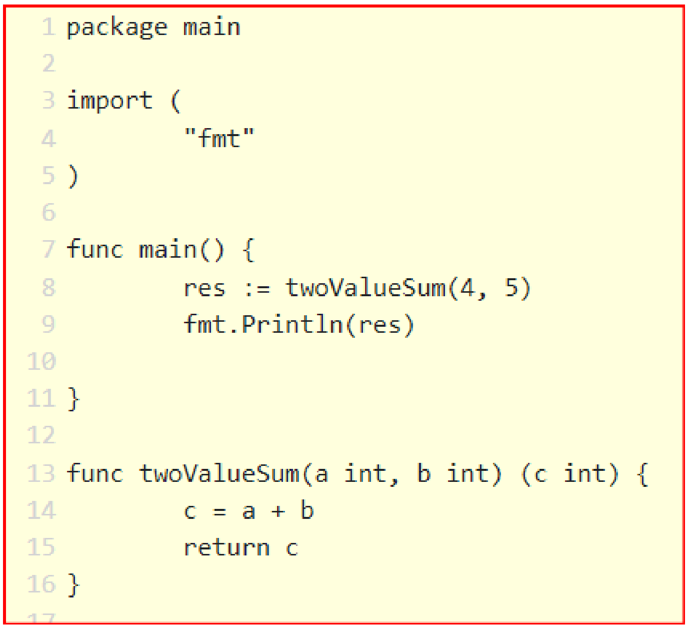
}

4.

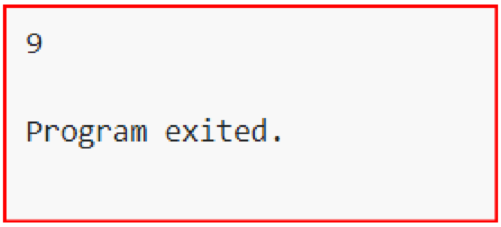
func <function_name>(<ipn> <ipt>, <ipn> <ipt>) (<rpn> <rpt>, <rpn> <rpt>) {

}

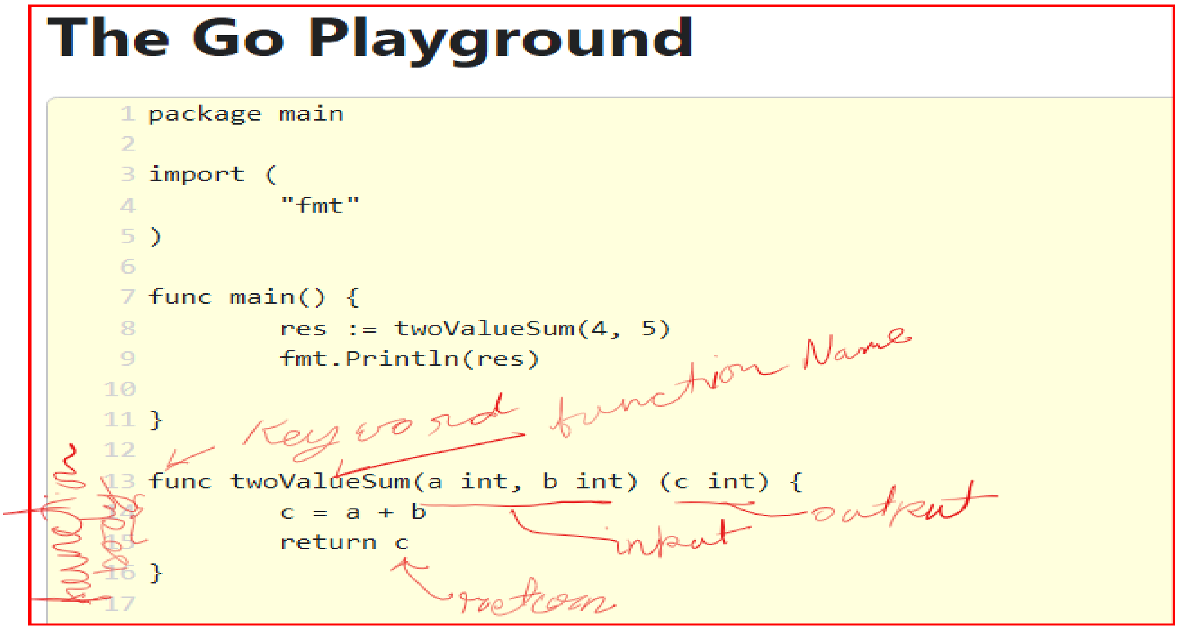
ipn => input_parameter_name
ipt => input _parameter_type
rpn => return_parameter_name
rpt => return_parameter_type



```go
package main

import (
        "fmt"
)

func main() {
        res := twoValueSum(4, 5)
        fmt.Println(res)

}

func twoValueSum(a int, b int) (c int) {
        c = a + b
        return c
}
```



```
9

Program exited.
```



The Go Playground

```
package main

import (
	"fmt"
)

func main() {
	res := twoValueSum(4, 5)
	fmt.Println(res)

}

func twoValueSum(a int, b int) (c int) {
	c = a + b
	return c
}

```

Keywords in Golang: -

break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	If	range	type
continue	for	import	return	var

append	bool	byte	cap	close	complex	complex64	complex128	uint16
copy	false	float32	float64	imag	int	int8	int16	uint32
int32	int64	iota	len	make	new	nil	panic	uint64
print	println	real	recover	string	true	uint	uint8	Uintptr

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

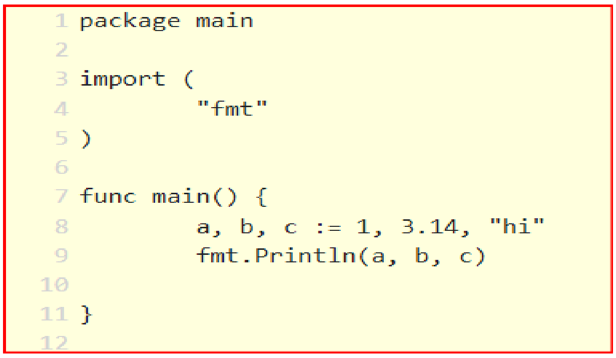
3.

<variable_name> := <value>

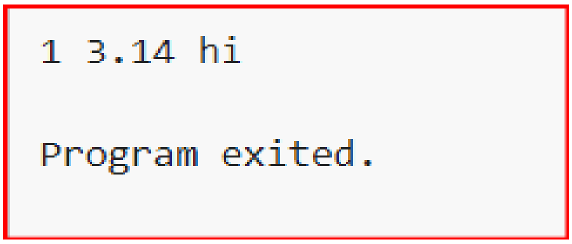
num1 := 4

num1,num2 :=4,5

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



```go
package main

import (
        "fmt"
)

func main() {
        a, b, c := 1, 3.14, "hi"
        fmt.Println(a, b, c)

}

```



```
1 3.14 hi

Program exited.
```

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 => &&, ||

2. *if else*

```
if <condition>{
} else{
}
```

3. *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{

}
```

Continue/ Break

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

Break is used to break out of loop.

Comments

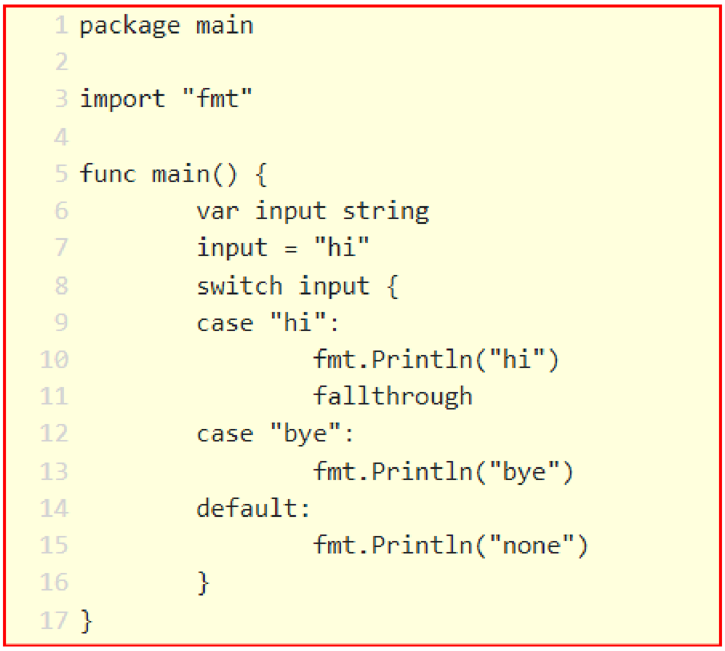
```
// single line comments
/* multi line
        Comments*/
```

Type Casting

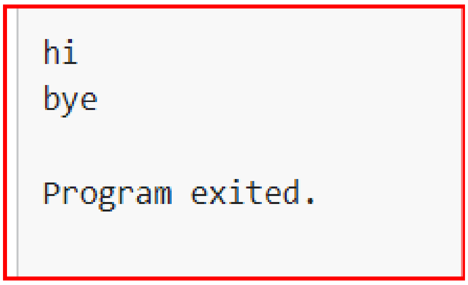
```
var <variable_name> <target_type>(<input_variable>)
val := 5.5
v := int(val)
```

Switch Case

```
switch (<input>){
case <val1>:
        fallthrough
case <val2>:

default:
```



```go
package main

import "fmt"

func main() {
        var input string
        input = "hi"
        switch input {
        case "hi":
                fmt.Println("hi")
                fallthrough
        case "bye":
                fmt.Println("bye")
        default:
                fmt.Println("none")
        }
}
```

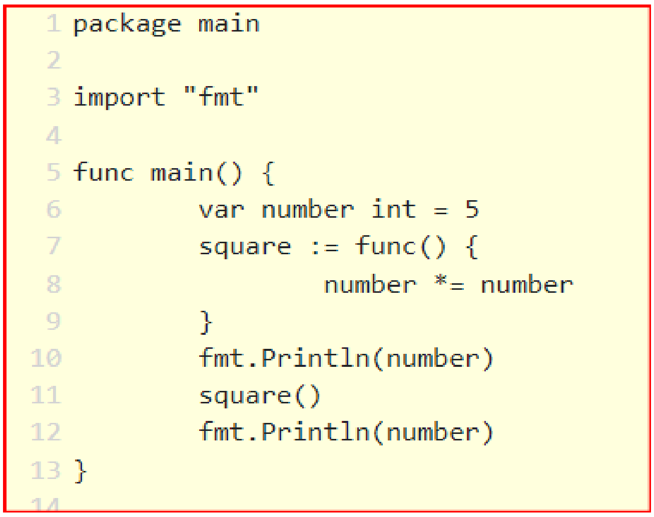


```
hi
bye

Program exited.
```

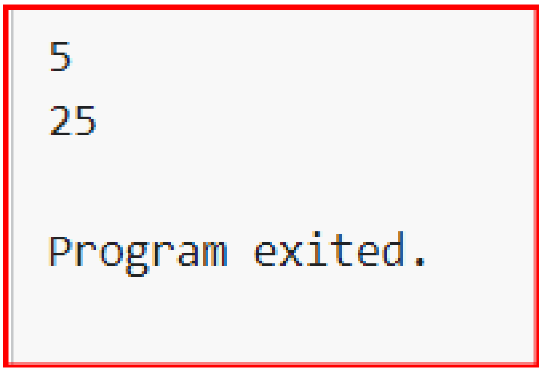
* falltrhough, is used to go in next case, as in Golang case default breaks.

Closure

closure is a function with similar property of a function which is declared inside another function



```go
package main

import "fmt"

func main() {
        var number int = 5
        square := func() {
                number *= number
        }
        fmt.Println(number)
        square()
        fmt.Println(number)
}
```



```
5
25


Program exited.
```

Array

var <variable_name> [<size>]<type>

<variable_name> := make([]<type> ,<size>)

Slice

Slice is dynamically sized array.

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>){
}
```

Structure

```
type <structure_name> struct {
<name> <type>
}
```

Embedded Structure

Structure inside structure is called embedded structure.

Rune

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

```
var <variable_name> rune = value
<variable_name> := ‘value’
```

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>)
```

Contacts

Name: - *Manindra Narayan Singh*

Email Id: - *mns.manindra@gmail.com*

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

***** Don’t call after 10 pm otherwise I will call back before 6 am.

* Mention your name in WhatsApp, Email or Message.

*