

## HOW TO GO!

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

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

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

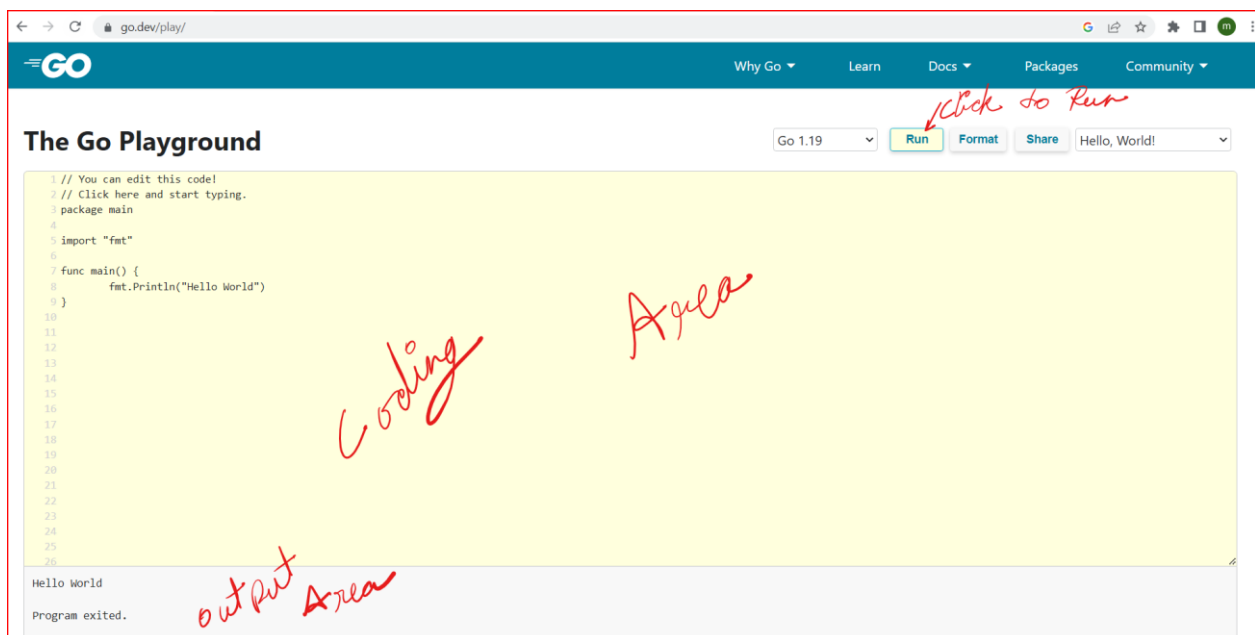
<https://qobyexample.com/>

### 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!"

```
1 package main  
2  
3 import "fmt"  
4  
5 func main() {  
6     fmt.Println("Hello World!")  
7 }  
8
```

Hello World!

## 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"  
        ...  
)
```

```
5 import    "fmt"  
6 import    "math"  
7 import("strconv"  
8         "strings"  
9 )
```

---

## 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

---

## What is function and how to write: -

A function is a group of statements that together perform a task.[\(link\)](#)

- **func** keyword is used to declare a function.

---

1.

```
func function_name( param_name param_type){  
}
```

Example: -

```
func sumTwoNum( a int, b int){  
    fmt.Println(a + b)  
}
```

---

2.

```
func function_name( param_name param_type) retron_type {  
}
```

Example: -

```
func sumTwoNum( a int, b int) int{  
    return a + b  
}
```

---

3.

```
func function_name( param_name param_type , param_name param_type ) ( retron_type ,  
retron_type) {  
}
```

Example: -

```
func sumDifferenceTwoNum( a int, b int) (int, int) {  
    return a + b, a - b  
}
```

---

4.

```
func function_name( param_name param_type , param_name param_type) (retrun_param_name  
retrun_type , retrun_param_name retrun_type) {  
  
}
```

Example: -

```
func sumDifferenceTwoNum( a int, b int) (c int, d int) {  
    c, d= a + b, a - b  
    return c, d  
}
```

```
1 package main  
2 import "fmt"  
3 func main() {  
4     sayHi("mani")  
5     fmt.Println(sum(3, 5))  
6     fmt.Println(multiply(3, 5))  
7     fmt.Println(quotRem(9, 4))  
8 }  
9 func sayHi(str string) {  
10     fmt.Println("hi", str)  
11 }  
12 func sum(a, b int) int {  
13     return a + b  
14 }  
15 func multiply(a int, b int) (c int) {  
16     c = a * b  
17     return c  
18 }  
19 func quotRem(a int, b int) (float64, int) {  
20     return float64(a) / float64(b), a % b  
21 }  
22
```

```
hi mani  
8  
15  
2.25 1
```

## Closure

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

```

1 package main
2
3 import "fmt"
4
5 func main() {
6     var number int = 5
7     square := func() {
8         number *= number
9     }
10    fmt.Println(number)
11    square()
12    fmt.Println(number)
13 }
14

```

5  
25

## 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 function_name ( variable_name ...type)(type){
}

```

```

7 func main() {
8     str := combineString("hello", "world")
9     fmt.Println(str)
10    str = combineString("hi", "how", "are", "you")
11    fmt.Println(str)
12 }
13 func combineString(strs ...string) string {
14     res := ""
15     for _, str := range strs {
16         res += str + " "
17     }
18     return res
19 }

```

hello world  
hi how are you

## Variable declaration in Golang: -

- **var** keyword is used to declare a variable.

1.

**var** *variable\_name* *variable\_type*

Example: -

**var** num1 int

**var** num1, num2 int

2.

**var** *variable\_name* *variable\_type* = *value*

Example: -

**var** num1,num2 int = 4,5

3.

*variable\_name* **:=** *value*

Example: -

num1 **:=** 4

num2,str1 **:=**4, "hi"

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

```
7 func main() {
8     var str1 string
9     str1 = "Welcome"
10
11     var num1, num2 int
12     num1, num2 = 5, 6
13
14     var str2, str3 string = "hi", "Mani"
15
16     num3, isTrue := num1+num2, true
17 }
```

## Constants

- **const** keyword is used to declare a constant.
- We don't use ":" in case of constant.

---

**const** *variable\_name* = value

**const** *variable\_name* *variable\_type* = value

Example: -

```
const c = 5
```

```
const d string = "hi"
```

---

```
7 func main() {  
8     const c = 5  
9     const d string = "hi"  
10 }  
11
```

---

## Rune

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

---

**var** *variable\_name* **rune** = value

*variable\_name* **:=** value

Example: -

```
Var numRune rune = '1'
```

```
charRune := 'a'
```

---

```
7 func main() {  
8     a, b := 'a', '2'  
9     fmt.Println(a, string(a), b, string(b))  
10 }  
11
```

```
97 a 50 2
```

---



## Array

`var variable_name [size] type`

`variable_name := make ( [ ] type , size)`

---

## Slice

Slice is dynamically sized array.

---

```
7 func main() {  
8     var arr1 [5]int  
9     arr1 = [5]int{1, 2, 3, 4}  
10    arr2 := [3]int{1, 2, 3}  
11    arr3 := arr1[1:3]  
12    fmt.Println(arr1, arr2, arr3)  
13 }  
14
```

```
[1 2 3 4 0] [1 2 3] [2 3]
```

---

## Controls

### if-else

#### 1. only if: -

```
if condition {  
}  
if condition1 logical_operator condition2 {  
}
```

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

Example: -

```
if a <= 5 {  
}  
  
If a == 5 && b ==6 {  
}
```

---

#### 2. if else : -

```
if condition {  
}  
else {  
}
```

Example: -

```
if a <= 5 {  
}  
else {  
}
```

---

#### 3. if else – ladder/chain: -

```
if condition {  
}  
else if condition {  
}  
else {  
}
```

Example: -

```
if a <= 5 {  
}  
else if a > 5 {  
}  
else {  
}
```

---

```

7 func main() {
8     a := 9
9     if a%10 == 0 && a >= 0 {
10         fmt.Println(a)
11     } else if a%2 == 0 && a >= 0 {
12         fmt.Println("even and +ve")
13     } else if a > 0 {
14         fmt.Println("odd and +ve")
15     } else {
16         fmt.Println("negative")
17     }
18 }
19

```

## 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

```

for index , value := range data_collection {
}

```

Example: -

```

Var values [5] int
for ind , value := range values {
}

```

```

7 func main() {
8     var names []string = []string{"Mani", "Animesh", "Shahul"}
9     // simple loop
10    for i := 0; i < len(names); i++ {
11        fmt.Print(names[i], " ")
12    }
13
14    // range
15    for index, name := range names {
16        fmt.Printf("%d %s,", index, name)
17    }
18 }
19

```

```

Mani
Animesh
Shahul

0 Mani
1 Animesh
2 Shahul

```

## Continue/ Break

Continue is used just like other language to skip to next iteration.

Break is used to break out of loop.

```

7 func main() {
8     for i := 0; i < 6; i++ {
9         if i%2 == 0 {
10             continue
11         }
12         fmt.Print(i, " ")
13     }
14     fmt.Println()
15     for i := 0; i < 6; i++ {
16         if i == 4 {
17             break
18         }
19         fmt.Print(i, " ")
20     }
21 }

```

```

1 3 5
0 1 2 3

```

## Comments

// single line comments

/\* multi line

Comments\*/

---

```
4
5 func main() {
6     //single line comment
7     /* multi
8     line
9     comments */
10 }
11
```

---

## Type Casting

**var** *variable\_name* *target\_type* ( *input\_variable* )

Example: -

value := 5.5

v := int ( value )

---

```
7 func main() {
8     var f float64 = 5.5
9     var a int
10    a = int(f)
11    fmt.Println(f, "&", a)
12 }
```

5.5 & 5

Program exited.

---

## Switch Case

```
switch (input){  
  case Value1:  
    fallthrough  
  case Value2:  
  default:
```

```
1 package main  
2  
3 import "fmt"  
4  
5 func main() {  
6     var input string  
7     input = "hi"  
8     switch input {  
9         case "hi":  
10            fmt.Println("hi")  
11            fallthrough  
12        case "bye":  
13            fmt.Println("bye")  
14        default:  
15            fmt.Println("none")  
16    }  
17 }
```

```
hi  
bye  
  
Program exited.
```

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

## Structure

- type and struct are the two key word used to define structure.

```
type structure_name struct {  
    field_name field_type  
}
```


Example: -

```
type Detail struct{  
    Name string  
}
```

## Embedded Structure

Structure inside structure is called embedded structure.

```
7 type PersonalInformation struct {
8     FName  string
9     LName  string
10    Mobile string
11    Addr   Address
12 }
13 type Address struct {
14     HouseNo int
15     Street  string
16     PinCode int
17 }
```



```
19 func main() {
20     person1 := PersonalInformation{
21         FName: "Mani",
22         LName: "Singh",
23         Addr: Address{
24             HouseNo: 204,
25         },
26     }
27     fmt.Println(person1)
28 }
```

```
{Mani Singh {204 0}}
```

Program exited.

## Goroutine

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** ()

```
10 func main() {  
11     var wg sync.WaitGroup  
12     wg.Add(2)  
13     go addNums(10, 0, &wg)  
14     go addNums(10, 2, &wg)  
15     wg.Wait()  
16 }  
17 func addNums(num1, num2 int, wg *sync.WaitGroup) {  
18     fmt.Println(num1 + num2)  
19     defer wg.Done()  
20 }
```

12

10

Program exited.

## 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](#))

- chan keyword is used to create channel.

### 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)**

Example: -

```
unbuffChan := make(chan string)
```

### Buffered channel

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

**variable\_name := make(chan variable\_type , channel\_size)**

Example: -

```
buffChan := make(chan string, 5)
```



```

7 func main() {
8     messageUnbuff := make(chan string) //unbuffered channel
9     messageBuff := make(chan string,2) //buffered channel
10
11     messageUnbuff <- "hi"    // send message in a channel
12     msg := <- messageUnbuff // receive message from channel
13
14 }

```

## Select

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

```

7 func main() {
8     messageUnbuff := make(chan string) //unbuffered channel
9     messageBuff := make(chan string,2) //buffered channel
10
11     messageUnbuff <- "hi"    // send message in a channel
12     msg := <- messageUnbuff // receive message from channel
13
14     select {
15     case msg1 := <-c1:
16         fmt.Println("received", msg1)
17     case msg2 := <-c2:
18         fmt.Println("received", msg2)
19     }
20 }

```

## 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 )**

Example: -

```
var newMap map [ string ] int
```

```
secondMap := make ( map [ string ] int )
```

```

7 func main() {
8     var mp map[string]int    //declarating
9     mp = make(map[string]int) //initializing
10
11     mp1 := make(map[string]int) //declarating & initializing
12
13     //declarating, initializing and assigning value
14     mp2 := map[string]int{
15         "hi":    2,
16         "hello": 5,
17     }
18 }

```

\* 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 **defer**, **panic** and **recover**.

---

```

10 func main() {
11     var err1 error
12     err1 = errors.New("this is a custom error")
13     str := "errorsss"
14     err2 := fmt.Errorf("this is second error %s", str)
15 }

```

---

## 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.

---

```
10 func main() {
11     fmt.Println(checkRecover(10, 0))
12     fmt.Println(checkRecover(10, 2))
13 }
14 func checkRecover(num1, num2 int) int {
15     defer func() {
16         fmt.Println(recover())
17     }()
18     if num2 == 0 {
19         panic(errors.New("Divide by zero"))
20     }
21     quotient := num1 / num2
22     return quotient
23 }
24
```

Divide by zero

0

<nil>

5

## Contacts

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

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

**LinkedIn:** - <https://www.linkedin.com/in/mani...>

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