

Slice and Maps no need to pass by reference in order to modify its value.

Pass by value is enough for this both to modify the original value in the called function.

All other data types other than this need to be pass by reference in-order to modify its original value by the function.

Ex: array, struct, int ,float,string,....

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Print("Hello")
7     fmt.Print("A")
8 }
9
```

```
HelloA
Program exited.
```

**Print** - Prints the output and there is no newline  
**Println** - Prints the output and there is newline

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello")
7     fmt.Print("A")
8 }
9
```

```
Hello
A
```

Go pkg installation  
This works

```
C:\Users\L\go>go get github.com/spf13/cobra
```

```
C:\Users\L\go>cd src
```

```
C:\Users\L\go\src>cd github.com
```

```
C:\Users\L\go\src\github.com>dir
```

```
Volume in drive C is Windows  
Volume Serial Number is 0C38-5DB2
```

```
Directory of C:\Users\L\go\src\github.com
```

```
19/03/2022  20:46    <DIR>          .  
19/03/2022  20:46    <DIR>          ..  
18/03/2022  07:53    <DIR>          google  
19/03/2022  20:46    <DIR>          inconsahreaveable  
19/03/2022  20:46    <DIR>          spf13  
                0 File(s)              0 bytes  
                5 Dir(s)  12,351,606,784 bytes free
```

```
C:\Users\L\go\src\github.com>cd spf13
```

```
C:\Users\L\go\src\github.com\spf13>dir
```

```
Volume in drive C is Windows  
Volume Serial Number is 0C38-5DB2
```

```
Directory of C:\Users\L\go\src\github.com\spf13
```

## Custom function for particular data type

```
5  type Person struct {
6  |   name  string
7  |   age   int
8  |   status string
9  | }
10
11 // func (receiver_type) func_name(){
12 func (p Person) print() {
13 |   fmt.Printf("Name :%s\n", p.name)
14 |   fmt.Printf("Age  :%d\n", p.age)
15 |   fmt.Printf("Status :%s\n", p.status)
16 | }
17
18 func main() {
19 |   me := Person{name: "Logesh", age: 23, status: "Unknown"}
20 |   me.print()
21 | }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs\receiver_function> go run .\working_with_receiver.go
Name :Logesh
Age :23
Status :Unknown
PS E:\_Golang\go_programs\receiver_function> █
```

This receiver function can be created only for the custom data types not for default data types

```
14     fm var some string
15     fm
16 }
17     View Problem   No quick fixes available
18 func (some string) print_addr() {
19     fmt.Printf("Address of the string is : %p", &some)
20 }
21
22 func main() {
23     // me := Person{name: "Logesh", age: 23, status: "Unknown"}
24     // me.print()
25     var s string = "lo"
26     s.print_addr()
27 }
28
```

So i created alias for the default datatype and implemented the receiver function for that data type

```
18  type custom_string string
19
20  func (some custom_string) print_addr() {
21  |   fmt.Printf("Address of the string is : %p", &some)
22  | }
23
24  func main() {
25  |   // me := Person{name: "Logesh", age: 23, status: "Unknown"}
26  |   // me.print()
27  |   var my_str custom_string
28  |   my_str = "The Logesh Vel"
29  |   my_str.print_addr()
30  | }
```

Created alias for the default data type and  
Created the receiver function

PROBLEMS

5

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\receiver_function> go run .\working_with_receiver.go
```

```
Address of the string is : 0xc000098220
```

```
PS E:\_Golang\go_programs\receiver_function> █
```

```
9 func main() {
10
11     newFile, err := os.Create("a.txt")
12     // error handling
13     if err != nil {
14         log.Fatal(err)
15     }
16 }
17 fmt.Fprintln(newFile, "written in program by Logesh!!!")
18 contents, err := os.ReadFile("a.txt")
19 if err != nil {
20     log.Fatal(err)
21 }
22 fmt.Println(contents)
23 fmt.Println(string(contents))
24 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

powerShell + ▢

```
PS E:\_Golang\go_programs\os_files> go run .\working_with_files.go
[119 114 105 116 116 101 110 32 105 110 32 112 114 111 103 114 97 109 32 98 121 32 76 111 103 101 115 104 33 33 33 10]
written in program by Logesh!!!
```

```
PS E:\_Golang\go_programs\os_files> █
```

Once readed its won't get the updated.

```
9 func main() {
10     new_file, err := os.Create("a.txt")
11     if err != nil {
12         fmt.Println(err)
13     }
14     fmt.Fprintln(new_file, "Logesh Vel")
15     created_file, err := os.ReadFile("a.txt")
16     if err != nil {
17         fmt.Println(err)
18     }
19     fmt.Println(string(created_file))
20     fmt.Fprintln(new_file, "Again Logesh Vel")
21     fmt.Println(string(created_file))
22
23 }
24
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\os_files> go run .\working_with_files.go
Logesh Vel
```

```
Logesh Vel
```

```
PS E:\_Golang\go_programs\os_files> █
```



```
9 func main() {
10     new_file, err := os.Create("a.txt")
11     if err != nil {
12         fmt.Println(err)
13     }
14     fmt.Fprintln(new_file, "Logesh Vel")
15     created_file, err := os.ReadFile("a.txt")
16     if err != nil {
17         fmt.Println(err)
18     }
19     fmt.Println(string(created_file))
20     fmt.Fprintln(new_file, "Again Logesh Vel")
21     created_file, err = os.ReadFile("a.txt")
22     _ = err
23     fmt.Println(string(created_file))
24 }
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\os_files> go run .\working_with_files.go
Logesh Vel
```

```
Logesh Vel
Again Logesh Vel
```

```
PS E:\_Golang\go_programs\os_files> █
```

```
10 func main() {  
11     data, err := ioutil.ReadFile("b.txt")  
12     _ = err  
13     fmt.Println(string(data))  
14 }
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

PS E:\\_Golang\go\_programs\os\_files\using\_ioutil> go run .\working\_with\_ioutil.go

I learn Golang! 传

PS E:\\_Golang\go\_programs\os\_files\using\_ioutil> █

## Scope of loop variables

```
12     for x := 1; 10 == 10; x++ {  
13         fmt.Println("a")  
14     }  
15     fmt.Println("outer:", x)  
16 }  
17
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\loop_vaariables> go run .\loop_var_scope.go  
# command-line-arguments  
.\loop_var_scope.go:15:24: undefined: x  
PS E:\_Golang\go_programs\loop_vaariables> █
```

# Variadic function parameters

If the **last parameter** of a function has type `...T`, it can be called with any number of trailing arguments of type `T`. The actual type of `...T` inside the function is `[]T`.

This example function can be called with, for instance, `Sum(1, 2, 3)` or `Sum()`.

```
func Sum(nums ...int) int {  
    res := 0  
    for _, n := range nums {  
        res += n  
    }  
    return res  
}
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello")
7     fmt.Println('A')
8     fmt.Println("A")
9 }
```

Hello

65

A

Program exited.

Single quotes takes exactly one character and prints the ASCII value of that character

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello %s", "s")
7     fmt.Printf("Hello %s", "s")
8 }
9
```

Hello %s s

Hello s

**Printf** - is the format print

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     s = "Man"
7     fmt.Printf("Hello %s", s)
8 }
9
```

```
./prog.go:6:2: undefined: s
./prog.go:7:25: undefined: s
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var s = "Man"
7     fmt.Printf("Hello %s", s)
8 }
9
```

```
Hello Man
```

## Short hand

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     s := "Man"
7     fmt.Printf("Hello %s", s)
8 }
9
```

Hello Man



```
1 | ///////////////////////////////////////////////////
2 | // The Typical Structure of a Go Application
3 | // Go Playground: https://play.golang.org/p/vY\_IeYBb1GN
4 | ///////////////////////////////////////////////////
5 |
6 | // a package clause starts every source file
7 | // main is a special name declaring an executable rather than a library (p
   | ackage)
8 | package main
9 |
10 | // import declaration declares packages used in this file
11 | import "fmt"
12 |
13 | // package scoped variables and constants
14 | var x int = 100
15 |
16 | const y = 0
17 |
18 | // a function declaration. main is a special function name
19 | // it is the entry point for the executable program
20 | // Go uses brace brackets to delimit code blocks
21 | func main() {
22 |
23 |     // Local Scoped Variables and Constants Declarations, calling function
   | s etc
24 |     var a int = 7
25 |     var b float64 = 3.5
26 |     const c int = 10
27 |
28 |     // Println() function prints out a line to stdout
29 |     // It belongs to package fmt
30 |     fmt.Println("Hello Go world!") // => Hello Go world!
31 |     fmt.Println(a, b, c)           // => 7 3.5 10
32 |
33 | }
```



**Good job!**

package main means an executable program.

Question 3:

Consider the following Go program that will be compiled.

Choose the correct statement regarding the outcome of the compilation process:

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

**This is an executable program.**

This is not an executable program. It's a package (library).



### Good job!

The main package must have package name main and declare a function main that takes **no arguments and returns no value.**

Question 4:

Having the following Go program, choose the correct statement:

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

**Func main() is mandatory for package main.**

**Func main() is optional.**



Good job!

Question 5:

Having the following Go program, what is `fmt` ?

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

- The name of a Go source file that is imported in the current file.
- The name of a Go Standard Library Package used mainly to print out messages at standard output (console).**
- The name of a Go Package that must be installed before imported. It's used to print messages.

**gofmt** - is used to format the file according to the Go standard.

Formatting is nothing but the aligning the codes in the file in the proper way

like **a=10** is formatted to **a = 10**

**gofmt -w filename.go** - format the file and writes back to the same file

**gofmt -w -l directory** - formats all the file in that directory

**go fmt** - is to format all the go files in that directory where this command is executed



Good job!

Question 1:

What is the difference between `go run` and `go build` ?

- `go run` and `go build` have the same purpose. Using specific options they can be used interchangeably.
- `go run` compiles and runs the application. It produces an executable that can be later run again.  
`go build` compiles and runs the application. It produces an executable that can be later run again.
- `go run` compiles and runs the application. It doesn't produce an executable.  
`go build` compiles but doesn't run the application. It should be run manually afterwards.



Good job!

Question 2:

You want to create an executable from your **main.go** file named **firewall\_automation.exe**.

What command should you execute?

go build --output firewall\_automation.exe

go build -o firewall\_automation.exe

go run -o firewall\_automation.exe



Good job!

This was discussed in Lecture 12: [Compiling \(go build\) and Running Go Applications \(go run\)](#) >

Question 3:

In a directory that contains many Go source files you run `go build`

What will happen?

- It will compile main.go and will produce an executable called main.exe
- It will compile all the source files in the current directory and will produce an executable with the name of the directory.**
- It will compile all the source files in the current directory and will produce an executable. It will also run the executable.





Good job!

Question 5:

You want to format all the Go source files in the current directory in the Go idiomatic way.

What command will you run?



`go fmt`

or

`gofmt -w .`



`gofmt`

or

`go fmt -w -l`



# Variables initialization

```
1 package main
2
3 import (
4     |   "fmt"
5 )
6
7 func main() {
8     |   var a int
9     |   fmt.Println(a)
10    |   var b int = 10
11    |   fmt.Println(b)
12    |   var c = 11
13    |   fmt.Println(c)
14    |   d := 10
15    |   fmt.Println(d)
16    |   // var e := 11 // missing variable type or initialization (var e invalid type)
17    |   // fmt.Println(e)
18 }
19
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS E:\\_Golang\go\_programs> go run .\first\_file.go

0

10

11

10

PS E:\\_Golang\go\_programs> █

```
package main

import (
    "fmt"
)

func main() {
    var a int
    fmt.Println(a)
    var b int = 10
    fmt.Println(b)
    var c = 11
    fmt.Println(c)
    d := 10
    fmt.Println(d)
    // var e := 11 // missing variable type or initialization (var e invalid type)
    // fmt.Println(e)
}
```

```
6
7 func main() {
8     var a []int
9     fmt.Println(a)
10    // var b = []int //expected expression
11    var b = []int{}
12    fmt.Println(b)
13    var c = []int{1, 3, 5, 6}
14    fmt.Println(c)
15    var d = make([]int, 3) // makes slice(list) of length 3
16    fmt.Println(d)
17    e := []int{18, 32, 53, 66}
18    fmt.Println(e)
19 }
20
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
[]
[]
[1 3 5 6]
[0 0 0]
[18 32 53 66]
PS E:\_Golang\go_programs> []
```

```
package main
```

```
import (  
    "fmt"  
)
```

```
func main() {  
    var a []int  
    fmt.Println(a)  
    // var b = []int //expected expression  
    var b = []int{}  
    fmt.Println(b)  
    var c = []int{1, 3, 5, 6}  
    fmt.Println(c)  
    var d = make([]int, 3) // makes slice(list) of length  
    3  
    fmt.Println(d)  
    e := []int{18, 32, 53, 66}  
    fmt.Println(e)  
}
```

```
7 func main() {
8     // array (fixed length)
9     var a [5]int
10    fmt.Println(a)
11    // var b = []int //expected expression
12    var b = [3]int{}
13    fmt.Println(b)
14    var c = [4]int{1, 3, 5, 6}
15    c[3] = 1
16    // c[4]=9 // invalid argument: index 4 (constant of type int) is out of bounds
17    fmt.Println(c)
18    var d = make([]int, 3) // makes array(list) of length 3
19    fmt.Println(d)
20    e := [5]int{18, 32, 53, 66}
21    fmt.Println(e)
22 }
23
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
[0 0 0 0 0]
[0 0 0]
[1 3 5 1]
[0 0 0]
[18 32 53 66 0]
PS E:\_Golang\go_programs> █
```

```
package main

import (
    "fmt"
)

func main() {
    // array (fixed length)
    var a [5]int
    fmt.Println(a)
    // var b = []int //expected expression
    var b = [3]int{}
    fmt.Println(b)
    var c = [4]int{1, 3, 5, 6}
    c[3] = 1
    // c[4]=9 // invalid argument: index 4 (constant of type int) is out of bounds
    fmt.Println(c)
    var d = make([]int, 3) // makes array(list) of length 3
    fmt.Println(d)
    e := [5]int{18, 32, 53, 66}
    fmt.Println(e)
}
```



```
7 func main() {
8     var a map[int]int
9     fmt.Println(a)
10    var b = map[int]int{0: 1, 1: 2}
11    fmt.Println(b)
12    var c = map[int]int{}
13    fmt.Println(c)
14    var d = make(map[int]int, 3) // makes map(dict)
15    d[1] = 1
16    fmt.Println(d)
17    e := map[int]int{}
18    e[0] = 0
19    fmt.Println(e)
20 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
map[]
```

```
map[0:1 1:2]
```

```
map[]
```

```
map[1:1]
```

```
map[0:0]
```

```
PS E:\_Golang\go_programs> █
```

```
package main

import (
    "fmt"
)

func main() {
    var a map[int]int
    fmt.Println(a)
    var b = map[int]int{0: 1, 1: 2}
    fmt.Println(b)
    var c = map[int]int{}
    fmt.Println(c)
    var d = make(map[int]int, 3) // makes map(dict)
    d[1] = 1
    fmt.Println(d)
    e := map[int]int{}
    e[0] = 0
    fmt.Println(e)
}
```

```
7
8  const s string = "constant"
9
10 func main() {
11     fmt.Println(s)
12
13     const n = 500000000
14
15     const d = 3e20 / n
16     fmt.Println(d)
17
18     fmt.Println(int64(d))
19
20     fmt.Println(math.Sin(n))
21 }
22
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
constant
6e+11
6000000000000
-0.28470407323754404
PS E:\_Golang\go_programs> █
```

```
package main

import (
    "fmt"
    "math"
)

const s string = "constant"

func main() {
    fmt.Println(s)

    const n = 500000000

    const d = 3e20 / n
    fmt.Println(d)
    fmt.Println(int64(d))
    fmt.Println(math.Sin(n))
}
```

# Function with Pointers parameters

```
6
7 func fun_pointers(a *int) {
8     fmt.Println("Address of given argument :", a)
9     fmt.Println("Value of the given argument:", *a)
10    fmt.Println("Changing the vlaue to 10")
11    *a = 10
12 }
13
14 func main() {
15     var to_change = 1
16     fmt.Println("Value of to_change before Pointer function call", to_change)
17     fun_pointers(&to_change)
18     fmt.Println("Value of to_change after Pointer function call", to_change)
19 }
20 }
21
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Value of to_change before Pointer function call 1
Address of given argument : 0xc0000aa058
Value of the given argument: 1
Changing the vlaue to 10
Value of to_change after Pointer function call 10
PS E:\_Golang\go_programs> █
```

```
package main

import (
    "fmt"
)

func fun_pointers(a *int) {
    fmt.Println("Address of given argument :", a)
    fmt.Println("Value of the given argument:", *a)
    fmt.Println("Changing the value to 10")
    *a = 10
}

func main() {
    var to_change = 1
    fmt.Println("Value of to_change before Pointer function call," to_change)
    fun_pointers(&to_change)
    fmt.Println("Value of to_change after Pointer function call," to_change)
}
```

We can pass by reference for all the data types but for the Maps we don't need to call function by pass by reference .

By default like in python (dict) pass by value is pass by reference for Maps.

So be careful while using the Maps(dicts) because they are passed as the reference by default so it may leads to unexpected altering the values.



```
6
7 func fun_pointers(a *[]int) {
8     fmt.Println("Address of given argument :", a)
9     fmt.Println("Value of the given argument:", *a)
10    fmt.Println("Changing the value of index 1 to 10")
11    (*a)[1] = 10 // *a[1]=10 gives error due to the precedence level. so first pointing and then indexing
12 }
13
14 func main() {
15     var to_change = []int{1, 2, 3}
16     fmt.Println("Value of to_change before Pointer function call", to_change)
17     fun_pointers(&to_change)
18     fmt.Println("Value of to_change after Pointer function call", to_change)
19 }
20 }
21
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Value of to_change before Pointer function call [1 2 3]
Address of given argument : &[amp;1 2 3]
Value of the given argument: [1 2 3]
Changing the value of index 1 to 10
Value of to_change after Pointer function call [1 10 3]
PS E:\_Golang\go_programs> █
```

```
package main
```

```
import (  
    "fmt"  
)
```

```
func fun_pointers(a *[]int) {  
    fmt.Println("Address of given argument :", a)  
    fmt.Println("Value of the given argument:", *a)  
    fmt.Println("Changing the value of index 1 to 10")  
    (*a)[1] = 10 // *a[1]=10 gives error due to the precedence level. so first pointing and then  
indexing  
}
```

```
func main() {  
    var to_change = []int{1, 2, 3}  
    fmt.Println("Value of to_change before Pointer function call," to_change)  
    fun_pointers(&to_change)  
    fmt.Println("Value of to_change after Pointer function call," to_change)  
}
```

## Change the value of Slice without the pointers in function call

```
1 package main
2
3 import "fmt"
4
5 func changer(a ...int) {
6     a[0] = 109
7     a[1] = 456
8 }
9
10 func main() {
11
12     my_slice := []int{1, 2, 3, 4, 5}
13     fmt.Println(my_slice)
14     changer(my_slice...)
15     fmt.Println(my_slice)
16 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL


```
PS E:\_Golang\go_programs\functions\ellipsis_parameter> go run .\function.go
[1 2 3 4 5]
[109 456 3 4 5]
PS E:\_Golang\go_programs\functions\ellipsis_parameter> █
```

```
10 func plus_changer(b int, others ...string) {
11     b = 100001
12     others[0] = "chnged"
13 }
14
15 func main() {
16
17     my_slice := []int{1, 2, 3, 4, 5}
18     fmt.Println(my_slice)
19     changer(my_slice...)
20     fmt.Println(my_slice)
21
22     my_strs := []string{"Logesh", "vel"}
23     my_var := 10
24     fmt.Println(my_var, my_strs)
25     plus_changer(my_var, my_strs...)
26     fmt.Println(my_var, my_strs)
27
28 }
```

PROBLEMS **1** OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\functions\ellipsis_parameter> go run .\function.go
[1 2 3 4 5]
[109 456 3 4 5]
10 [Logesh vel]
10 [chnged vel]
PS E:\_Golang\go_programs\functions\ellipsis_parameter> []
```

# Function with return values

```
go first_file.go >  vals
1  package main
2
3  import "fmt"
4
5  func vals() (int, int) {
6      return 3, 7
7  }
8
9  func main() {
10
11     a, b := vals()
12     fmt.Println(a, b)
13
14     _, c := vals()
15     fmt.Println(c)
16 }
17
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
3 7
```

```
7
```

```
PS E:\_Golang\go_programs> █
```

```
package main

import "fmt"

func vals() (int, int) {
    return 3, 7
}

func main() {

    a, b := vals()
    fmt.Println(a, b)

    _, c := vals()
    fmt.Println(c)

}
```

## Variables in Go.

Variables must be declared before use and the declared variable must be used

Ex:

```
var a int = 10
```

```
var s string
```

```
s = "Logesh"
```

```
B := "Hey" this is the shorthand to define the variable
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello buddy")
7     s = "Logesh"
8 }
9
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
# command-line-arguments
.\first_file.go:7:2: undefined: s
PS E:\_Golang\go_programs>
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello buddy")
7     var s = "Logesh"
8 }
9
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
# command-line-arguments
.\first_file.go:7:6: s declared but not used
PS E:\_Golang\go_programs>
```



```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hello buddy")
7     var s = "Logesh"
8     fmt.Printf(s)
9 }
10
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Hello buddy
Logesh
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var s string = "Logesh"
7     fmt.Printf(s)
8
9     s := "re-assign"
10
11 }
12
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
# command-line-arguments
.\first_file.go:9:4: no new variables on left side of :=
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var s string = "Logesh"
7     fmt.Println(s)
8     s = "re-assigned"
9     fmt.Println(s)
10
11 }
12
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Logesh
re-assigned
PS E:\_Golang\go_programs> █
```

**:= is only for declaring the new variables**

## Working with constants

```
go first_file.go > main
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const c = 100
7     fmt.Println(c)
8     const cc int = 10
9     fmt.Println(cc)
10 }
11
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
100
```

```
10
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const c = 100
7     fmt.Println(c)
8     c = 1
9 }
10
```

PROBLEMS **1** OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
# command-line-arguments
```

```
.\first_file.go:8:4: cannot assign to c (declared const)
```

```
PS E:\_Golang\go_programs> █
```

If the situation arises to work with the unused variables. Use `_` to suppress the error

```
1 package main
2
3 func main() {
4     var _ = 10
5 }
6
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
PS E:\_Golang\go_programs> █
```

## Multiple declarations

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     a, b, c := 1, 2, 3
7     var A, B, C = 4, 5, 6
8     fmt.Println(a, b, c)
9     fmt.Println(A, B, C)
10
11 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
1 2 3
```

```
4 5 6
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var A, B, C int = 4, 5, 6
7     fmt.Println(A, B, C)
8
9 }
10
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
4 5 6
```

```
PS E:\_Golang\go_programs> █
```

We can redeclare the variable using multiple declaration and short hand combinations

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var A, B, C int = 4, 5, 6
7     fmt.Println(A, B, C)
8     A := 3
9 }
10
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
# command-line-arguments
.\first_file.go:8:4: no new variables on left side of :=
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var A, B, C int = 4, 5, 6
7     fmt.Println(A, B, C)
8     A, a := 3, 10
9     fmt.Println(A, B, C, a)
10
11 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
4 5 6
3 5 6 10
PS E:\_Golang\go_programs> █
```

Now the **A** is redeclared using **:=** along with the new variable with it.

## Another way of multiple declaration without assigning the value and **with assigning the value**

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var a, b, c int
7     fmt.Println(a, b, c)
8     var (
9         is_user    bool
10        user_name string
11        age         int
12    )
13    fmt.Println(is_user, user_name, age)
14 }
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
0 0 0
false 0
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var a, b, c int = 10, 10, 2
7     fmt.Println(a, b, c)
8     var (
9         is_user    bool    = true
10        user_name string = "Logesh"
11        age         int     = 21
12    )
13    fmt.Println(is_user, user_name, age)
14 }
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
10 10 2
true Logesh 21
PS E:\_Golang\go_programs> █
```

cannot assign 1 values to 3 variables compiler([WrongAssignCount](#))

[View Problem](#) No quick fixes available

```
test, test1, test2 = 5
```





Good job!

Question 2:

You want to declare variable called `balance` that stores a `float64` value.

Choose the right way:

`balance := 12.5` or `var balance float64 = 12.5` or `var balance = 12.5`

`balance float := 12.5` or `var balance float64 = 12.5`

`var balance := 12.5`



Good job!

Question 3:

Having the following Go program, how is `_` called and what is its purpose?

```
1 | package main
2 |
3 | import "fmt"
4 |
5 | func main() {
6 |     fmt.Println("Hello Go World!")
7 |     x := 10
8 |     _ = x
9 | }
```

`_` is the blank identifier and is used to avoid the error of declared but unused variables.

`_` is the blank identifier and is used to delete a variable.

`_` is the underscore identifier and is used to flush (zero value) a variable.



**Good job!**

Question 5:

Can we omit the type when declaring a variable using the normal declaration syntax like `var x = 10` ?

No. Go is a statically, strong type programming language.

Yes. The type will be inferred.



Good job!

Question 6:

Having the following Go program, will it compile without error?

```
1 | package main
2 |
3 | import "fmt"
4 |
5 | y := 20
6 |
7 | func main() {
8 |     x := 10
9 |     fmt.Println(x)
10 | }
```

Yes.

No. `y` is declared but not used.

No. You cannot use short-declaration (`:=`) for package scoped variables.

In Go all variables are initialized. Even if we haven't gave the value Go will assign the default value

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     var s string
8     var i int
9     var f float32
10    var b bool
11    fmt.Println(s, i, f, b)
12
13 }
14
```

String - empty string ""

Numeric - 0

Bool - false

Pointer - nil

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
0 0 false
```

```
PS E:\_Golang\go_programs> █
```

```
/** ZERO VALUES **/
```

```
    // An uninitialized variable or empty variable will get the so  
called ZERO VALUE
```

```
    // The zero-value mechanism of Go ensures that a variable always  
holds a well defined value of its type
```

```
    var value int           // initialized with 0  
    var price float64      // initialized with 0.0  
    var name string        // initialized with empty  
string -> ""  
    var done bool          // initialized with false  
    fmt.Println(value, price, name, done) // -> 0 0.0 "" false
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var x float64
7     fmt.Println(x, x == 0.0)
8 }
9 |
```

0 true

```

// VERBS:
// %d -> decimal
// %f -> float
// %s -> string
// %q -> double-quoted string
// %v -> value (any)
// %#v -> a Go-syntax representation of the value
// %T -> value Type
// %t -> bool (true or false)
// %p -> pointer (address in base 16, with leading 0x)
// %c -> char (rune) represented by the corresponding Unicode code point

a, b, c := 10, 15.5, "Gophers"
grades := []int{10, 20, 30}

fmt.Printf("a is %d, b is %f, c is %s \n" , a, b, c)    // => a is 10, b is 15.500000, c is
Gophers
fmt.Printf("%q\n", c)                                // => "Gophers"
fmt.Printf("%v\n", grades)                            // => [10 20 30]
fmt.Printf("%#v\n", grades)                          // => b is of type float64 and grades is of type
[]int
fmt.Printf("b is of type %T and grades is of type %T\n" , b, grades)
// => b is of type float64 and grades is of type []int
fmt.Printf("The address of a: %p\n" , &a)            // => The address of a: 0xc000016128
fmt.Printf("%c and %c\n", 100, 51011)                // => d and 𐄀 (runes for code points 101 and
51011)

```



```
const pi float64 = 3.14159265359
```

```
fmt.Printf("pi is %.4f\n", pi) // => formatting with 4 decimal points
```

```
// %b -> base 2
```

```
// %x -> base 16
```

```
fmt.Printf("255 in base 2 is %b\n", 255) // => 255 in base 2 is 11111111
```

```
fmt.Printf("101 in base 16 is %x\n", 101) // => 101 in base 16 is 65
```

```
// fmt.Sprintf() returns a string. Uses the same verbs as fmt.Printf()
```

```
s := fmt.Sprintf("a is %d, b is %f, c is %s \n", a, b, c)
```

```
fmt.Println(s) // => a is 10, b is 15.500000, c is Gophers
```

```
fmt.Printf("255 in base 2 is %08b\n", 5) //255 in base 2 is 00000101
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     |
7     |   fmt.Printf("255 in base 2 is %b\n", 5)
8     |   fmt.Printf("255 in base 2 is %08b\n", 5)
9     |
10    | }
11
```

---

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
255 in base 2 is 101
255 in base 2 is 00000101
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     grades := []int{10, 20, 30}
8     fmt.Printf("%#v\n", grades)
9     fmt.Printf("%#v\n", grades[0])
10
11 }
```

---

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
[]int{10, 20, 30}
10
PS E:\_Golang\go_programs> █
```



Good job!

Question 5:

You want to print out the result of  $1.3 * 4.5$  with **3 decimal points**. How can you do it?



1

| `fmt.Printf("%3f\n", 1.3*4.5)`



1

| `fmt.Printf("%.3f\n", 1.3*4.5)`



1

| `fmt.Printf("%f[3]\n", 1.3*4.5)`

```
1 package main
2
3 func main() {
4     const year int = 2022
5 }
6
```

---

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
PS E:\_Golang\go_programs> █
```

No error is thrown . unused year.

Only the **variables** will get the error unused variable

**Const** must be initialized at the time of declaration

For **variables** it is not necessary it has the default value (Zero values)

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     //grouped const
7     const (
8         a int    = 1
9         b string = "Logesh"
10        c
11    ) // here the c will get the type and value from the previous value
12    // this is the behaviour of the grouped const
13    fmt.Println(a, b, c)
14 }
15
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
1 Logesh Logesh
```

```
PS E:\_Golang\go_programs> []
```

## Typed and Untyped Constant

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const x = 5 // untyped const (becoz the type is not mentioned)
7     const y = 1.7 * x // untyped const (becoz the type is not mentioned)
8     fmt.Printf("Type of x: %T\n", x)
9     fmt.Printf("Type of y: %T\n", y)
10
11 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

powerShell + v

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
Type of x: int
```

```
Type of y: float64
```

```
PS E:\_Golang\go_programs> █
```

Same code as previous but produces error. Its all due to mentioning the type so we can't use with other types.(Strong typed)

```
1  package main
2
3  import "fmt"
4
5  func main() {
6      const x int = 5           // typed const (becoz the type is mentioned)
7      const y float64 = 1.7 * x // typed const (becoz the type is mentioned)
8      fmt.Printf("Type of x: %T\n", x)
9      fmt.Printf("Type of y: %T\n", y)
10
11 }
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

powerShell + v

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
# command-line-arguments
```

```
.\first_file.go:7:8: cannot use 1.7 * x (type int) as type float64 in const initializer
```

```
.\first_file.go:7:24: constant 1.7 truncated to integer
```

```
PS E:\_Golang\go_programs> █
```



```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const (
7         min1 = -500
8         max1 //gets its type and value form the previous constant. It's -500
9         max2 //in a grouped constants, a constant repeats the previous one -> -500
10    )
11    fmt.Println(min1, max1, max2)
12 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
-500 -500 -500
```

```
PS E:\_Golang\go_programs> █
```

```
// Declaring multiple (grouped) constants
const (
    a          = 5    // untyped constant
    b float64 = 0.1 // typed constant
)
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const (
7         c1 = iota
8         c2 = iota
9         c3 = iota
10    )
11    fmt.Println(c1, c2, c3) // => 0 1 2
12    const (
13        North = iota //by default 0
14        East      //omitting type and value means, repeating its type and value so East = iota = 1(it increments by 1 automatically)
15        South     // -> 2
16        West      // -> 3
17    )
18    fmt.Println(North, East, South, West)
19 }
20
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

powerShell + v

```
PS E:\_Golang\go_programs> go run .\first_file.go
0 1 2
0 1 2 3
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const (
7         c1 = iota - 5.9
8         c2
9         c3
10    )
11    fmt.Println(c1, c2, c3)
12 }
13
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
-5.9 -4.9 -3.9
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const ( // << and >> are the binary shift operators
7         c1 = 0b0001 << 1 // left shift.means shift all the bits towards left by one bit as we gave 1
8         c2
9         c3
10    )
11    fmt.Printf("%04b , %04b , %04b\n", c1, c2, c3)
12    fmt.Printf("%T , %T , %T", c1, c2, c3)
13
14 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
0010 , 0010 , 0010
int , int , int
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     const ( // << and >> are the binary shift operators
7         c1 = 0b0001 >> 1 // right shift.means shift all the bits towards right by one bit as we gave 1
8         c2
9         c3
10    )
11    fmt.Printf("%04b , %04b , %04b\n", c1, c2, c3)
12    fmt.Printf("%T , %T , %T", c1, c2, c3)
13
14 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
0000 , 0000 , 0000
int , int , int
PS E:\_Golang\go_programs> █
```



**Good job!**

x is an untyped constant and it will get its type from the first expression where it's used.

Question 2:

What will the following Go program print out ?

```
1 | package main
2 |
3 | import "fmt"
4 |
5 | func main() {
6 |     const x = 7
7 |     const y float64 = 3.1
8 |     fmt.Println(x * y)
9 | }
```



There is an error. You cannot multiply an int (x) by a float (y) in Go.



21.7



### Good job!

There are ONLY boolean constants, rune constants, integer constants, floating-point constants, complex constants, and string constants.

Question 6:

After declaring an array like `var x = [2]int{1, 2}`, you want to create a **constant** of type **array** and write `const y = [2]int{5, 6}`

However, you get an error. What is the problem and the possible solution?

You cannot declare constants of type array.

You should write: `const y [2]int = {5, 6}`



```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // rune is character type holds only one character in the single quotes
7     var r1 rune = "a"
8     fmt.Println(r1)
9 }
10
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
# command-line-arguments
```

```
.\first_file.go:7:6: cannot use "a" (type untyped string) as type rune in assignment
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // rune is character type holds only one character in the single quotes
7     var r1 rune = 'a'
8     fmt.Println(r1) // prints the ASCII value for that character
9     fmt.Printf("%T", r1)
10 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
97
```

```
int32
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // array - fixed length with the items of same type
7     var r1 = [5]int{1, 5, 10, 49, 3}
8     fmt.Println(r1)
9     fmt.Printf("%T", r1)
10 }
11
```

---

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
[1 5 10 49 3]
```

```
[5]int
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // slice - dynamic length with the items of same type
7     var r1 = []int{1, 5, 10, 49, 3}
8     fmt.Println(r1)
9     fmt.Printf("%T", r1)
10 }
11
```

---

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
[1 5 10 49 3]
```

```
[]int
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // map - dict in python - but in Go all keys should be of same type and all vlaues should of tha same type
7     var m1 = map[string]int{
8         "Logesh": 21,
9     } // syntax - map[key_data_type]values_datatype{}
10    fmt.Println(m1["Logesh"])
11    fmt.Println(m1)
12    fmt.Printf("%T", m1)
13 }
14
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
21
```

```
map[Logesh:21]
```

```
map[string]int
```

```
PS E:\_Golang\go_programs> █
```

## User defined data type - **struct**

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     type my_type struct {
7         name string
8         age int
9     }
10    var me my_type
11    fmt.Println(me)
12    fmt.Printf("%T\n", me)
13    me.name = "Logesh"
14    me.age = 21
15    fmt.Println(me)
16    fmt.Println(me.name)
17 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
{ 0}
main.my_type
{Logesh 21}
Logesh
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // Pointer type
7     var x = 1
8     fmt.Println(x)
9     var ptr = &x // &var - gives address, *ptr - gives value stored in that address, ptr - gives the address of that var
10    fmt.Printf("Pointer type - %T\nValue of ptr - %v\nValue in that addr : %v", ptr, ptr, *ptr)
11
12 }
13
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
1
```

```
Pointer type - *int
```

```
Value of ptr - 0xc000014098
```

```
Value in that addr : 1
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // Function Type
7     fmt.Printf("Function type - %T", function_1)
8 }
9
10 func function_1() {
11     fmt.Println("Called function_1")
12 }
13
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
Function type - func()
```

```
PS E:\_Golang\go_programs> █
```





Good job!

Question 2:

The following Go program returns an error. Why is that?

```
1 | package main
2 |
3 | func main() {
4 |     var distance uint8
5 |     distance = 5 * 100
6 |     _ = distance
7 | }
```

distance is declared but not used.

**5 \* 100 overflows uint8**

package main imported and not used.



Good job!

Question 3:

There is an error in the following program. Your task is to make the program to compile.

What would you do?

```
1 | package main
2 |
3 | import "fmt"
4 |
5 | func main() {
6 |     var golang string = 'Go'
7 |     fmt.Println(golang)
8 | }
9 |
```



It's not possible to use go for a variable name. It's a language keyword.



Replace single quotes ( ' ') with double quotes(" ") when declaring the string.



Good job!

Question 4:

Which variable is of type **slice**?

1 | `x := []int{}`

2 | `y := [5]int{}`

**x**

**y**

**Neither of them.**



Good job!

Question 5:

What is the type of **p**?

1 | x := 1

2 | p := &x

int

pointer to int

channel

rune

Question 1:

What will the following Go program print out?

```
1 | package main
2 |
3 | import "fmt"
4 |
5 | func main() {
6 |     var v rune
7 |     fmt.Printf("%T\n", v)
8 | }
```

rune

int32

## String concatenation

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     a := "Logesh"
7     fmt.Println(a + " vel")
8 }
9
```

---

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_GOlang\go_programs> go run .\first_file.go
```

```
Logesh vel
```

```
PS E:\_GOlang\go_programs> █
```

# Operators in Go

---

An **operator** is a symbol of the programming language which is able to **operate on values**.

In Go language, operators can be categorized based upon their different functionality in these categories:

- **Arithmetic and Bitwise Operators:** +, -, \*, /, %, &, |, ^, <<, >>
- **Assignment Operators:** +=, -=, \*=, /=, %=
- **Increment and Decrement Statements:** ++, --
- **Comparison Operators:** ==, !=, <, >, <=, >=
- **Logical Operators:** &&, ||, !
- **Operators for Pointers (&) and Channels (<-)**

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // OVERFLOW
7     var a uint8 = 255 // max value that can be stored
8     fmt.Println(a)
9     a++ // 256 which is overflowed so (256 - overflowed_value) here 256 is the total value that uint8 can be stored
10    fmt.Println(a)
11    a++ // 257 which is overflowed so (256 - overflowed_value)
12    fmt.Println(a)
13 }
14
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
255
```

```
0
```

```
1
```

```
PS E:\_Golang\go_programs> █
```



Go not allow to initialize the overflowed value.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // OVERFLOW
7     var a uint8 = 256 // uint can store ( 0 to 255 )
8     fmt.Println(a)
9 }
10
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
# command-line-arguments
.\first_file.go:7:6: constant 256 overflows uint8
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     var a = string(65) // string of int value will return the corresponding charcater that has that ASCII
7     fmt.Println(a)
8     fmt.Printf("Type of a - %T\n", a)
9     b := fmt.Sprintf("%d", 65) // Sprintf converts the int 65 to String 65
10    fmt.Println(b)
11    fmt.Printf("Type of b - %T", b)
12
13 }
14
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
A
```

```
Type of a - string
```

```
65
```

```
Type of b - string
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import (
4     "fmt"
5     "strconv"
6 )
7
8 func main() {
9     // STRING to Number conversion
10    a := "1.34"
11    fmt.Printf("Type of a - %T : Value of a - %v\n", a, a)
12    f, err := strconv.ParseFloat(a, 64) // parameters 1-string to convert 2-bitsize of the result float conversion (32 or 64)
13    fmt.Printf("Type of f - %T : Value of f - %v\n", f, f)
14    fmt.Println("The Possible error during the conversion ", err)
15 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

powershell

```
PS E:\_Golang\go_programs> go run .\first_file.go
Type of a - string : Value of a - 1.34
Type of f - float64 : Value of f - 1.34
The Possible error during the conversion <nil>
PS E:\_Golang\go_programs> █
```

Another most used conversion is **Atoi()** , **Itoa()**

**ASCII to Int**

**Int to ASCII**

```
1 package main
2
3 import (
4     "fmt"
5     "strconv"
6 )
7
8 func main() {
9     // STRING to Number conversion (Atoi and Itoa)
10    s := "-100"
11    fmt.Printf("Type of s - %T : Value of s - %v\n", s, s)
12    i, err := strconv.Atoi(s) // string that has the int
13    fmt.Printf("Type of i - %T : Value of i - %v\n", i, i)
14    fmt.Println("The Possible error during the conversion ", err)
15 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Type of s - string : Value of s - -100
Type of i - int : Value of i - -100
The Possible error during the conversion <nil>
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import (
4     "fmt"
5     "strconv"
6 )
7
8 func main() {
9     // Number to STRING conversion (Itoa)
10    i := 65
11    fmt.Printf("Type of i - %T : Value of i - %v\n", i, i)
12    s := strconv.Itoa(i)
13    fmt.Printf("Type of s - %T : Value of s - %v\n", s, s)
14 }
15
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
Type of i - int : Value of i - 65
```

```
Type of s - string : Value of s - 65
```

```
PS E:\_Golang\go_programs> █
```



**Good job!**

Use `math.Pow()` to calculate the power.

This was discussed in Lecture 34: [Operations on Types: Arithmetic and Assignment Operators](#) >

Question 1:

Which is power (exponentiation) operator in Go?

\*\*

^

**There is no power operator in Go.**



Good job!

Question 5:

What will be the type of **x**?

```
1 | var x = fmt.Sprintf("%d", 34234)
```

**string**

int

decimal

float64

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     // Defined types
9     type my_type int // we can create our new type based on the existing type as its underlying type
10    var m_t my_type
11    fmt.Printf("Type of m_t - %T\n", m_t)
12    fmt.Printf("Value of m_t - %v", m_t)
13 }
14
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Type of m_t - main.my_type
Value of m_t - 0
PS E:\_Golang\go_programs> █
```



```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     // Alias. Unlike Defined type Alias not create new type. It just creates alias for that type
9     type my_type = int // now my_type is same as of int (can be used inplace for int as alias)
10    var m_t my_type
11    fmt.Printf("Type of m_t - %T\n", m_t)
12    fmt.Printf("Value of m_t - %v\n", m_t)
13 }
14
```

---

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Type of m_t - int
Value of m_t - 0
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     var a uint8 = 10 // declaring a variable of type uint8
9     var b byte      // byte is an alias to uint8
10    // even though they have different names, byte and uint8 are the same type because they are aliases
11    fmt.Printf("Type of a - %T\n", a)
12    fmt.Printf("Type of b - %T\n", b)
13 }
14
```

---

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Type of a - uint8
Type of b - uint8
PS E:\_Golang\go_programs> █
```



Good job!

Question 2:

Having the following program, what is the **underlying type of duration**?

```
1 | package main
2 |
3 | import "fmt"
4 |
5 | type second uint
6 | type duration second
7 |
8 | func main() {
9 |     var d duration
10 |    fmt.Printf("%T\n", d)
11 | }
```



second



uint



Good job!

Question 3:

Having the following Go program, choose the correct statement:

```
1 package main
2
3 import "fmt"
4
5 type second uint
6 type duration second
7
8 type s = uint
9
10 func main() {
11 }
```

`s` is an alias to `uint`  
`second` is a named type

`s` is a named type  
`second` is an alias

Both `s` and `second` are named types or aliases for `uint`.





Good job!

Question 4:

Consider the following Go program.

Does the program compile without errors?

```
1 | package main
2 |
3 | type second uint
4 | type duration second
5 |
6 | type minute = uint
7 |
8 | func main() {
9 |     var t1 duration = 10
10 |
11 |     var x uint = t1
12 |     _ = x
13 | }
```



There are errors. You cannot assign type duration to type uint.

`var x uint = t1` is an error.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     var a []int
8     fmt.Println("emp:", a) // emp: []
9     for i := 0; i < 10; i++ {
10         a = append(a, i+10)
11
12     }
13     fmt.Println("emp:", a) //emp: [10 11 12 13 14 15 16 17 18 19]
14
15 }
16
```

changer &gt; changer.go &gt; map\_changer

```
4
5 func changer(list []int) {
6     for i := 0; i < 10; i++ {
7         list = append(list, i+1)
8     }
9     fmt.Println("List in function ", list)
10 }
11
12
13 func map_changer(dict map[int]int) {
14     for i := 0; i < 10; i++ {
15         dict[i] = i + 1
16     }
17     fmt.Println("Dict in function", dict)
18 }
19
20
21 func main() {
22     // list is not modified when we pass by value
23     var a []int
24     fmt.Println("emp:", a)
25     changer(a)
26     fmt.Println("emp:", a)
27
28     // map is modified even we pass by value
29     b := make(map[int]int)
30     fmt.Println("Dict ", b)
31     map_changer(b)
32     fmt.Println("Dict ", b)
33 }
34
```

```
PS E:\_Golang\go_programs\changer> go run .\changer.go
emp: []
List in function [1 2 3 4 5 6 7 8 9 10]
emp: []
Dict map[]
Dict in function map[0:1 1:2 2:3 3:4 4:5 5:6 6:7 7:8 8:9 9:10]
Dict map[0:1 1:2 2:3 3:4 4:5 5:6 6:7 7:8 8:9 9:10]
PS E:\_Golang\go_programs\changer>
```

```
changer > -GO changer.go > ...
```

```
1 package main
2
3 import "fmt"
4
5 func changer(list *[]int) {
6     for i := 0; i < 10; i++ {
7         *list = append(*list, i+1)
8     }
9 }
10 fmt.Println("List in function ", *list)
11 }
12
13 func main() {
14     // list is modified when we pass by reference (address)
15     var a []int
16     fmt.Println("emp:", a)
17     changer(&a)
18     fmt.Println("emp:", a)
19
20     // we don't need pass by reference for map
21     // pass by value behaves the same
22
23 }
24
```

```
PS E:\_Golang\go_programs\changer> go run .\changer.go
emp: []
List in function [1 2 3 4 5 6 7 8 9 10]
emp: [1 2 3 4 5 6 7 8 9 10]
PS E:\_Golang\go_programs\changer> █
```



**Go routines**

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // go routines
7     go func(msg string) {
8         fmt.Println(msg)
9     }("going")
10 }
11
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // normal anonymous function
7     func(msg string) {
8         fmt.Println(msg)
9     }("going")
10 }
11
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
going
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import (
4     fmt
5     time
6 )
7
8 func main() {
9     // go routines
10    go func(msg string) {
11        fmt.Println(msg)
12    }("going")
13    time.Sleep(1) // one secs
14 }
15
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
```

```
going
```

```
PS E:\_Golang\go_programs> █
```

```
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func main() {
9     fmt.Println(time.Second)
10 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PS E:\\_Golang\go\_programs> go run .\first\_file.go

1s

PS E:\\_Golang\go\_programs> █

# Anonymous Function

```
2
3 import (
4     |   "fmt"
5 )
6
7 func main() {
8
9     func() {
10        |   fmt.Println("Its anonymous")
11    }()
12
13    func(msg string) {
14        |   fmt.Println(msg)
15    }("Its msg")
16 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\anonymous_func> go run .\anonymous.go
```

```
Its anonymous
```

```
Its msg
```

```
PS E:\_Golang\go_programs\anonymous_func> █
```

**Go Packets**

```
1 package main
2
3 import (
4     "fmt"
5
6     "github.com/google/gopacket/pcap"
7 )
8
9 func main() {
10     version := pcap.Version()
11     fmt.Println(version)
12 }
```

PROBLEMS

3

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs> go get github.com/google/gopacket/pcap
go: downloading github.com/google/gopacket v1.1.19
go: downloading golang.org/x/sys v0.0.0-20190412213103-97732733099d
PS E:\_Golang\go_programs> ls
```



To install the pkg open the cmd as Administrator permission and  
Go to the path : **C:\Program Files\Go\src**  
And issue the **go get github.com/google/gopacket**

```
C:\Program Files\Go\src>go get github.com/google/gopacket
go get: added github.com/google/gopacket v1.1.19

C:\Program Files\Go\src>_
```

```
1 package main
2
3 import (
4     "fmt"
5
6     "github.com/google/gopacket/pcap"
7 )
8
9 func main() {
10     version := pcap.Version()
11     fmt.Println(version)
12 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
first_file.go:6:2: no required module provides package github.com/google/gopacket/pcap: go.mod file not found in current directory or any parent d
irectory; see 'go help modules'
PS E:\_Golang\go_programs> go env -w GO111MODULE=off
PS E:\_Golang\go_programs> go run .\first_file.go
Npcap version 1.00, based on libpcap version 1.9.1
PS E:\_Golang\go_programs> █
```

```
PS E:\_GOlang\go_programs> go run .\first_file.go
```

```
first_file.go:6:2: no required module provides package github.com/google/gopacket/pcap: go.mod file not found in current directory or any parent directory; see 'go help modules'
```

```
PS E:\_GOlang\go_programs> go env -w GO111MODULE=off
```

```
PS E:\_GOlang\go_programs> go run .\first_file.go
```

```
Npcap version 1.00, based on libpcap version 1.9.1
```

```
PS E:\_GOlang\go_programs>
```

```
package main
import (
    "fmt"
    "github.com/google/gopacket"
    "github.com/google/gopacket/pcap"
)
func main() {
    version := pcap.Version()
    fmt.Println(version)
    o_handle, _ := pcap.OpenOffline("DNS_pcap.pcapng")
    defer o_handle.Close()
    packetsource := gopacket.NewPacketSource(o_handle, o_handle.LinkType(),)
    pkt, _ := packetsource.NextPacket()
    // fmt.Println(pkt)
    fmt.Printf("pkt type %T", pkt)
    fmt.Println(pkt.Metadata().CaptureInfo)
    fmt.Printf("Type of pkt.Metadata().CaptureInfo is %T\n", pkt.Metadata().CaptureInfo)
    aa := pkt.Metadata().CaptureInfo.Timestamp.Unix()
    fmt.Println(aa)
}
```

```

22     defer o_handle.Close()
23
24     packetsource := gopacket.NewPacketSource(
25         o_handle,
26         o_handle.LinkType(),
27     )
28     pkt, _ := packetsource.NextPacket()
29     // fmt.Println(pkt)
30     fmt.Printf("pkt type %T", pkt)
31     fmt.Println(pkt.Metadata().CaptureInfo)
32     fmt.Printf("Type of pkt.Metadata().CaptureInfo is %T\n", pkt.Metadata().CaptureInfo)
33     aa := pkt.Metadata().CaptureInfo.Timestamp.Unix()
34     fmt.Println(aa)
35 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

PS E:\_Golang\go_programs> go run .\first_file.go
Npcap version 1.00, based on libpcap version 1.9.1
Rann
pkt type *gopacket.eagerPacket{2021-12-19 20:23:27.962903 +0530 IST 90 90 0 []}
Type of pkt.Metadata().CaptureInfo is gopacket.CaptureInfo
1639925607
PS E:\_Golang\go_programs> 

```

```

.go
2022-03-18 08:5
-62135596800
PS E:\_Golang\g
.go
2022-03-18 09:0
-62135596800
PS E:\_Golang\g
.go
2022-03-18 09:0

```

```
28     pkt, _ := packetsource.NextPacket()
29     // fmt.Println(pkt)
30     fmt.Printf("pkt type %T", pkt)
31     fmt.Println(pkt.Metadata().CaptureInfo)
32     fmt.Printf("Type of pkt.Metadata().CaptureInfo is %T\n", pkt.Metadata().CaptureInfo)
33     aa := pkt.Metadata().CaptureInfo.Timestamp
34     fmt.Println(aa.Unix())
35     fmt.Println(aa.UTC())
36     fmt.Println(aa.Local())
37
38 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs> go run .\first_file.go
Npcap version 1.00, based on libpcap version 1.9.1
Rann
pkt type *gopacket.eagerPacket{2021-12-19 20:23:27.962903 +0530 IST 90 90 0 []}
Type of pkt.Metadata().CaptureInfo is gopacket.CaptureInfo
1639925607
2021-12-19 14:53:27.962903 +0000 UTC
2021-12-19 20:23:27.962903 +0530 IST
PS E:\_Golang\go_programs> █
```

```
.go
2022-03-18 08:58
-62135596800
PS E:\_Golang\go.
.go
2022-03-18 09:01
-62135596800
PS E:\_Golang\go.
.go
2022-03-18 09:03
```

# Channels

```
5     /
6
7     func main() {
8
9         messages := make(chan string)
10
11        go func() { messages <- "ping" }()
12        msg := <-messages
13        fmt.Println(msg)
14        go func() { messages <- "traceroute" }()
15        msg = <-messages
16        fmt.Println(msg)
17
18    }
19
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go
ping
traceroute
PS E:\_Golang\go_programs\channels> █
```

```
package main

import (
    "fmt"
)

func main() {
    messages := make(chan string)
    go func() { messages <- "ping" }()
    msg := <-messages
    fmt.Println(msg)
    go func() { messages <- "traceroute" }()
    msg = <-messages
    fmt.Println(msg)
}
```

*Channels* are the pipes that connect concurrent goroutines. You can send values into channels from one goroutine and receive those values into another goroutine.

*Send* a value into a channel using the `channel <-` syntax.

The `<-channel` syntax *receives* a value from the channel.



In Unbuffered channel we can send value then we can receive the value only once. When we try to receive the value from the channel which has no value in it or it has delivered already then we will get an error

Channel can store value but once the value is received from the channel the subsequent call to receive value from channel will raise error.

Channel is like queue but not the queue. We can override the value in the channel by the subsequent send to channel. But can't receive subsequently.

The receive concept of the channel will coincide with the queue concept. Once the value is got from the queue we can't again get from that queue.

```
4
5 func main() {
6     // unbuffered channel
7     messages := make(chan string)
8
9     go func() {
10        messages <- "buffered"
11    }()
12    msg := <-messages
13    fmt.Println(msg)
14    new_msg := <-messages
15    fmt.Println(new_msg)
16 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go
buffered
fatal error: all goroutines are asleep - deadlock!
```

```
goroutine 1 [chan receive]:
main.main()
    E:/_Golang/go_programs/channels/channels.go:14 +0xe5
exit status 2
PS E:\_Golang\go_programs\channels> █
```

```
5 func main() {
6     // unbuffered channel
7     messages := make(chan string)
8     // send value to channel
9     go func() {
10        messages <- "buffered"
11    }()
12    // receive the value from the channel. now the channel is empty after receive
13    msg := <-messages
14    fmt.Println(msg)
15    // now the channel is empty sending new value
16    go func() {
17        messages <- "new buffered"
18    }()
19    // now the channel has new value in it so we are receiving it
20    new_msg := <-messages
21    fmt.Println(new_msg)
22 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go
buffered
new buffered
PS E:\_Golang\go_programs\channels> █
```

```
5 func main() {
6     // unbuffered channel
7     messages := make(chan string)
8     // send value to channel
9     go func() {
10        |   messages <- "value"
11        | }()
12        // receive the value from the channel. now the channel is empty after receive
13        msg := <-messages
14        fmt.Println(msg)
15        // now the channel is empty sending new value
16        go func() {
17            |   messages <- "new value"
18            | }()
19            // override the value in that channel
20            go func() {
21                |   messages <- "new new value"
22                | }()
23            // now the channel has new value in it so we are receiving it
24            new_msg := <-messages
25            fmt.Println(new_msg)
26        }
27    }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go
value
new new value
PS E:\_Golang\go_programs\channels> █
```

```
package main
import "fmt"
func main() {
    // unbuffered channel
    messages := make(chan string)
    // send value to channel
    go func() { messages <- "value" }()
    // receive the value from the channel. now the channel is empty after receive
    msg := <-messages
    fmt.Println(msg)
    // now the channel is empty sending new value
    go func() { messages <- "new value" }()
    // override the value in that channel
    go func() { messages <- "new new value" }()
    // now the channel has new value in it so we are receiving it
    new_msg := <-messages
    fmt.Println(new_msg)
}
```

**Buffered channel**

By default channels are *unbuffered*, meaning that they will only accept sends (`chan <-`) if there is a corresponding receive (`<- chan`) ready to receive the sent value. *Buffered channels* accept a limited number of values without a corresponding receiver for those values.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     messages := make(chan string, 2)
8
9     messages <- "buffered"
10    messages <- "channel"
11
12    fmt.Println(<-messages)
13    fmt.Println(<-messages)
14 }
15
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go
buffered
channel
PS E:\_Golang\go_programs\channels> █
```

```
package main

import "fmt"

func main() {

    messages := make(chan string, 2)

    messages <- "buffered"
    messages <- "channel"

    fmt.Println(<-messages)
    fmt.Println(<-messages)

}
```

Here we got an error,  
Becoz the channel has the  
capacity to store 2 values but  
now it has only one value.  
At the line 12 we have  
received that value , now the  
channel has no value but at  
the next line we are trying to  
receive the value that doesn't  
exists.

```
5 func main() {  
6  
7     messages := make(chan string, 2)  
8  
9     messages <- "buffered"  
10    // messages <- "channel"  
11  
12    fmt.Println(<-messages)  
13    fmt.Println(<-messages)  
14 }  
15
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go  
buffered
```

```
fatal error: all goroutines are asleep - deadlock!
```

```
goroutine 1 [chan receive]:
```

```
main.main()
```

```
    E:/_Golang/go_programs/channels/channels.go:13 +0xaa
```

```
exit status 2
```

```
PS E:\_Golang\go_programs\channels> █
```

```
8 func worker(done chan bool) {
9     fmt.Print("working...")
10    time.Sleep(time.Second)
11    // here we are sending the value to the channel
12    done <- true
13    time.Sleep(time.Second)
14    fmt.Println("done updating")
15 }
16
17 func main() {
18
19     done := make(chan bool, 1)
20     go worker(done) // making go routine
21     // by default whenever the channel is used in another go routine and
22     // we try to receive the value from that channel it will wait until that go routine sends value to that channel
23     // once the channel gets the value the other go routine which waits to receive that value will receive and continues
24     <-done
25     // here the done upating will never print in the terminal becoz once the channel has the value the receive will
26     // receive the value and break that wait and conitues here it exits.
27 }
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go
```

```
working...
```

```
PS E:\_Golang\go_programs\channels> █
```



```
8 func worker(done chan bool) {
9     fmt.Print("working...")
10    time.Sleep(time.Duration(2))
11    fmt.Println("done updating")
12    // here we are sending the value to the channel
13    done <- true
14 }
15
16 func main() {
17     done := make(chan bool, 1)
18     go worker(done) // making go routine
19     // by default whenever the channel is used in another go routine and
20     // we try to receive the value from that channel it will wait until that go routine sends value to that channel
21     // once the channel gets the value the other go routine which waits to receive that value will receive and continues
22     <-done
23 }
24
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go
working...done updating
PS E:\_Golang\go_programs\channels> █
```

```
 8 func worker(done chan bool) {
 9     fmt.Print("working...")
10     time.Sleep(time.Duration(2))
11     fmt.Println("done updating")
12     // here we are sending the value to the channel
13     done <- true
14 }
15
16 func main() {
17     done := make(chan bool, 1)
18     go worker(done) // making go routine
19     // by default whenever the channel is used in another go routine and
20     // we try to receive the value from that channel it will wait until that go routine sends value to that channel
21     // once the channel gets the value the other go routine which waits to receive that value will receive and continues
22     // <-done
23     // what if we haven't try to receive the value from the channel that is used by one go routine.
24     // it won't wait.. Becoz its not going to receive so it won't wait.
25     // so we can't see that prints from the worker go routine functions becoz our main func won't wait.
26 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS E:\\_Golang\go\_programs\channels> go run .\channels.go

PS E:\\_Golang\go\_programs\channels> █

# Channel Directions

When using channels as function parameters, you can specify if a channel is meant to only send or receive values. This specificity increases the type-safety of the program.

```
1 package main
2
3 import "fmt"
4
5 func ping(pings chan<- string, msg string) {
6     pings <- msg
7 }
8
9 func pong(pings <-chan string, pongs chan<- string) {
10     msg := <-pings
11     pongs <- msg
12 }
13
14 func main() {
15     pings := make(chan string, 1)
16     pongs := make(chan string, 1)
17     ping(pings, "passed message")
18     pong(pings, pongs)
19     fmt.Println(<-pongs)
20 }
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\channels> go run .\channels.go
passed message
PS E:\_Golang\go_programs\channels> █
```

# Timeouts

```
8 func main() {
9
10     c1 := make(chan string, 1)
11     go func() {
12         time.Sleep(2 * time.Second)
13         c1 <- "result 1"
14     }()
15
16     select {
17     case res := <-c1:
18         fmt.Println(res)
19     case <-time.After(1 * time.Second):
20         fmt.Println("timeout 1")
21     }
22
23     c2 := make(chan string, 1)
24     go func() {
25         time.Sleep(2 * time.Second)
26         c2 <- "result 2"
27     }()
28     select {
29     case res := <-c2:
30         fmt.Println(res)
31     case <-time.After(3 * time.Second):
32         fmt.Println("timeout 2")
33     }
34 }
35
```

```
PS E:\_Golang\go_programs\timeouts>
PS E:\_Golang\go_programs\timeouts>
PS E:\_Golang\go_programs\timeouts> go run .\timeouts.go
timeout 1
result 2
PS E:\_Golang\go_programs\timeouts> █
```

For our example, suppose we're executing an external call that returns its result on a channel `c1` after 2s. Note that the channel is buffered, so the send in the goroutine is nonblocking. This is a common pattern to prevent goroutine leaks in case the channel is never read.

Here's the `select` implementing a timeout. `res := <-c1` awaits the result and `<-time.After` awaits a value to be sent after the timeout of 1s. Since `select` proceeds with the first receive that's ready, we'll take the timeout case if the operation takes more than the allowed 1s.

If we allow a longer timeout of 3s, then the receive from `c2` will succeed and we'll print the result.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     queue := make(chan string, 2)
8     queue <- "one"
9     queue <- "two"
10    close(queue)
11    // closed the channel
12    for elem := range queue {
13        fmt.Println(elem)
14    }
15 }
16
```

```
PS E:\_Golang\go_programs\channels>
PS E:\_Golang\go_programs\channels> go run .\channels.go
one
two
PS E:\_Golang\go_programs\channels> █
```

## Range over Channels

This `range` iterates over each element as it's received from `queue`. Because we `closed` the channel above, the iteration terminates after receiving the 2 elements.

This example also showed that it's possible to close a non-empty channel but still have the remaining values be received.



```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     queue := make(chan string, 2)
8     queue <- "one"
9     queue <- "two"
10    // close(queue)
11
12    for elem := range queue {
13        |   fmt.Println(elem)
14    }
15 }
16
```

```
PS E:\_Golang\go_programs\channels>
PS E:\_Golang\go_programs\channels> go run .\channels.go
one
two
fatal error: all goroutines are asleep - deadlock!

goroutine 1 [chan receive]:
main.main()
    E:/_Golang/go_programs/channels/channels.go:12 +0xae
exit status 2
PS E:\_Golang\go_programs\channels> █
```

We got error when we try to loop (range) through the channel beyond the value it stored then we got the error. But we can solve this error by closing the channel.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6
7     queue := make(chan string, 2)
8     queue <- "one"
9     queue <- "two"
10    close(queue)
11
12    for elem := range queue {
13        fmt.Println(elem)
14    }
15    fmt.Println("Sending new value to the channel")
16    queue <- "three"
17 }
18
```

```
PS E:\_Golang\go_programs\channels>
PS E:\_Golang\go_programs\channels> go run .\channels.go
one
two
Sending new value to the channel
panic: send on closed channel

goroutine 1 [running]:
main.main()
    E:/_Golang/go_programs/channels/channels.go:16 +0x10b
exit status 2
PS E:\_Golang\go_programs\channels> █
```

We can receive the value from the closed channel until the buffer values. But we can't send values to the closed channel

```
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func main() {
9
10     timer1 := time.NewTimer(2 * time.Second)
11
12     <-timer1.C
13     fmt.Println("Timer 1 fired")
14 }
15
```

```
PS E:\_Golang\go_programs\timers>
PS E:\_Golang\go_programs\timers> go run .\timers.go
Timer 1 fired
PS E:\_Golang\go_programs\timers> █
```

```
1 package main
2
3 import (
4     "fmt"
5     "time"
6 )
7
8 func main() {
9     ticker := time.NewTicker(500 * time.Millisecond)
10    done := make(chan bool)
11    go func() {
12        for {
13            select {
14            case <-done:
15                fmt.Println("Got value for the done")
16                return
17            case t := <-ticker.C:
18                fmt.Println("Tick at", t)
19            }
20        }
21    }()
22    time.Sleep(1600 * time.Millisecond)
23    ticker.Stop()
24    done <- true
25    fmt.Println("Ticker stopped")
26    time.Sleep(time.Duration(1))
27    // to see the print statement in go routines
28 }
29
```

```
PS E:\_Golang\go_programs\tickers>
PS E:\_Golang\go_programs\tickers> go run .\tickers_in_go.go
Tick at 2022-03-19 15:54:03.855996 +0530 IST m=+0.517920501
Tick at 2022-03-19 15:54:04.3653718 +0530 IST m=+1.027296301
Tick at 2022-03-19 15:54:04.8510283 +0530 IST m=+1.512952801
Ticker stopped
Got value for the done
PS E:\_Golang\go_programs\tickers> █
```

# WaitGroups

```
5     "sync"
6     "time"
7 )
8
9 func worker(id int) {
10     fmt.Printf("Worker %d starting\n", id)
11     time.Sleep(time.Second)
12     fmt.Printf("Worker %d done\n", id)
13 }
14
15 func main() {
16     var wg sync.WaitGroup
17
18     for i := 1; i <= 5; i++ {
19         wg.Add(1)
20
21         i := i
22
23         go func() {
24             defer wg.Done()
25             worker(i)
26         }()
27     }
28
29     wg.Wait()
30
31 }
32
```

```
PS E:\_Golang\go_programs\waitgrps> go run .\waitGroups_in_go.go
Worker 1 starting
Worker 2 starting
Worker 5 starting
Worker 4 starting
Worker 3 starting
Worker 3 done
Worker 4 done
Worker 1 done
Worker 2 done
Worker 5 done
PS E:\_Golang\go_programs\waitgrps>
```

To wait for multiple goroutines to finish, we can use a *wait group*.

Launch several goroutines and increment the WaitGroup counter for each.

```
wg.Add(1)
```

Block until the WaitGroup counter goes back to 0; all the workers notified they're done.

```
9 func worker(id int) {
10     fmt.Printf("Worker %d starting\n", id)
11     time.Sleep(time.Second)
12     fmt.Printf("Worker %d done\n", id)
13 }
14
15 func main() {
16     var wg sync.WaitGroup
17     wg.Add(3) // only 3
18     for i := 1; i <= 5; i++ {
19         i := i
20         go func() {
21             defer wg.Done()
22             worker(i)
23         }()
24     }
25     wg.Wait()
26 }
27
```

```
PS E:\_Golang\go_programs\waitgrps>
PS E:\_Golang\go_programs\waitgrps> go run .\waitGroups_in_go.go
Worker 5 starting
Worker 3 starting
Worker 4 starting
Worker 2 starting
Worker 1 starting
Worker 1 done
Worker 3 done
Worker 5 done
PS E:\_Golang\go_programs\waitgrps>
```

Here we have given `wg.Add(3)` so it will wait for 3 concurrent goroutines to finish

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     for i := 1; i <= 5; i++ {
7         fmt.Println("Loop var i=", i)
8         i := i + 2
9         fmt.Println("Inside Loop var i=", i)
10    }
11 }
12
```

```
PS E:\_Golang\go_programs\loop_vaariables>
PS E:\_Golang\go_programs\loop_vaariables> go run .\loop_var_scope.go
Loop var i= 1
Inside Loop var i= 3
Loop var i= 2
Inside Loop var i= 4
Loop var i= 3
Inside Loop var i= 5
Loop var i= 4
Inside Loop var i= 6
Loop var i= 5
Inside Loop var i= 7
PS E:\_Golang\go_programs\loop_vaariables> □
```



```
1 package main
2
3 import (
4     "fmt"
5     "os"
6 )
7
8 func main() {
9
10     // panic("a problem")
11     _, err := os.Create("E:\\_Golang\\go_programs\\panic\\file.txt")
12
13     if err != nil {
14         panic(err)
15     }
16     fmt.Println("Successfully created the file")
17 }
18
```

```
PS E:\_Golang\go_programs\panic>
PS E:\_Golang\go_programs\panic> go run .\panic_in_go.go
Successfully created the file
PS E:\_Golang\go_programs\panic> █
```

```
8 func main() {
9
10     f := createFile("E:\\_Golang\\go_programs\\defers\\defer.txt")
11     defer closeFile(f)
12     writeFile(f)
13 }
14
15 func createFile(p string) *os.File {
16     fmt.Println("creating")
17     f, err := os.Create(p)
18     if err != nil {
19         panic(err)
20     }
21     return f
22 }
23
24 func writeFile(f *os.File) {
25     fmt.Println("writing")
26     fmt.Fprintln(f, "data")
27 }
28
29 func closeFile(f *os.File) {
30     fmt.Println("closing")
31     err := f.Close()
32     if err != nil {
33         fmt.Fprintf(os.Stderr, "error: %v\n", err)
34         os.Exit(1)
35     }
36 }
37
```

```
PS E:\_Golang\go_programs\defers>
PS E:\_Golang\go_programs\defers> go run .\defer_in_go.go
creating
writing
closing
PS E:\_Golang\go_programs\defers> █
```

# Defer

*Defer* is used to ensure that a function call is performed later in a program's execution, usually for purposes of cleanup. `defer` is often used where e.g. `ensure` and `finally` would be used in other languages.

Immediately after getting a file object with `createFile`, we defer the closing of that file with `closeFile`. This will be executed at the end of the enclosing function (`main`), after `writeFile` has finished.

`Sprintf` formats and returns a string without printing it anywhere.

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Sprintln("only strings")
7 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\string_wrks> go run .\strings_in_go.go
PS E:\_Golang\go_programs\string_wrks> █
```

```
3 import "fmt"
4
5 func main() {
6     s := fmt.Sprintln("only strings")
7     fmt.Println(s)
8 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\string_wrks> go run .\strings_in_go.go
only strings
```

```
PS E:\_Golang\go_programs\string_wrks> █
```

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6     "os"
7 )
8
9 func main() {
10
11     mapD := map[string]int{"one": 5, "two": 7}
12     mapB, _ := json.Marshal(mapD)
13     fmt.Println(string(mapB))
14
15     enc := json.NewEncoder(os.Stdout)
16     d := map[string]int{"apple": 5, "lettuce": 7}
17     enc.SetIndent("", " ")
18     enc.Encode(d)
19     enc.Encode(mapD)
20 }
21
```

```
PS E:\_Golang\go_programs\go_json>
PS E:\_Golang\go_programs\go_json> go run .\json.go
{"one":5,"two":7}
{
  "apple": 5,
  "lettuce": 7
}
{
  "one": 5,
  "two": 7
}
PS E:\_Golang\go_programs\go_json> █
```

```
package main

import (
    "encoding/json"
    "fmt"
    "os"
)

func main() {

    mapD := map[string]int{"one": 5, "two": 7}
    mapB, _ := json.Marshal(mapD)
    fmt.Println(string(mapB))

    enc := json.NewEncoder(os.Stdout)
    d := map[string]int{"apple": 5, "lettuce": 7}
    enc.SetIndent("", " ")
    enc.Encode(d)
    enc.Encode(mapD)
}
```

Files

files > go files\_in\_go.go > main

```
2
3 import (
4     "fmt"
5     "os"
6 )
7
8 func check(e error) {
9     if e != nil {
10        panic(e)
11    }
12 }
13
14 func main() {
15
16     dat, err := os.ReadFile("E:\\_Golang\\go_programs\\files\\dummy.txt")
17     check(err)
18     fmt.Println(string(dat))
19     for l := range dat {
20         fmt.Println(l)
21     }
22 }
```



files > dummy.txt

1 data

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\files> go run .\files_in_go.go
data
0
1
2
3
PS E:\_Golang\go_programs\files> █
```



files > go files\_in\_go.go > main

```
4     "fmt"
5     "os"
6 )
7
8 func check(e error) {
9     if e != nil {
10        panic(e)
11    }
12 }
13
14 func main() {
15
16     dat, err := os.ReadFile("E:\\_Golang\\go_programs\\files\\dummy.txt")
17     check(err)
18     fmt.Println(string(dat))
19     for ln, l := range dat {
20         fmt.Println("Line No:", ln, "ASCII :", l, "Character :", string(l))
21     }
22 }
```

files > dummy.txt

1 data

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS E:\\_Golang\go\_programs\files> go run .\files\_in\_go.go

data

Line No: 0 ASCII : 100 Character : d

Line No: 1 ASCII : 97 Character : a

Line No: 2 ASCII : 116 Character : t

Line No: 3 ASCII : 97 Character : a

PS E:\\_Golang\go\_programs\files> █

# Labels

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     a := 2
7     fmt.Println("a is", a)
8     come_here:
9     if a <= 5 {
10        fmt.Println("a is(", a, ") less than 5 so incrementing it")
11        a++
12        goto come_here
13    } else {
14        fmt.Println("a(", a, ") is greater than 5")
15    }
16 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\labels> go run .\working_with_labels.go
a is 2
a is( 2 ) less than 5 so incrementing it
a is( 3 ) less than 5 so incrementing it
a is( 4 ) less than 5 so incrementing it
a is( 5 ) less than 5 so incrementing it
a( 6 ) is greater than 5
PS E:\_Golang\go_programs\labels> █
```

labels > -go working\_with\_labels.go > main

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     a := 2
7     fmt.Println("a is", a)
8     come_here:
9     if a <= 5 {
10        fmt.Println("a is(", a, ") less than 5 so incrementing it")
11        a++
12        goto come_here
13    } else {
14        fmt.Println("a(", a, ") is greater than 5")
15        goto go_here
16    }
17
18    fmt.Println("outer")
19
20    go_here:
21        fmt.Println("Finally in go here")
22
23 }
24
```

```
PS E:\_Golang\go_programs\labels> go run .\working_with_labels.go
a is 2
a is( 2 ) less than 5 so incrementing it
a is( 3 ) less than 5 so incrementing it
a is( 4 ) less than 5 so incrementing it
a is( 5 ) less than 5 so incrementing it
a( 6 ) is greater than 5
Finally in go here
PS E:\_Golang\go_programs\labels>
```

```
5 func main() {
6     a := 2
7     fmt.Println("a is", a)
8     come_here:
9     if a <= 5 {
10        fmt.Println("a is(", a, ") less than 5 so incrementing it")
11        a++
12        goto come_here
13    } else {
14        fmt.Println("a(", a, ") is greater than 5")
15        goto go_here
16    }
17    unreachable code unreachable
18
19    View Problem Quick Fix... (Ctrl+.)
20    fmt.Println("outer")
21
22    go_here:
23        fmt.Println("Finally in go here")
24
25 }
26
```



```
// There are 3 Scopes:
// - File Scope
// - Package Scope
// - Block (local) Scope

package main
// import statements are file scoped
import (
    "fmt"
    // import "fmt" -> error, within the same scope, unique names
    // importing as another name (alias) is permitted
    f "fmt"
)
// variables or constant declared outside any function are package scoped
const done = false

func main() { // package scoped

    // block scoped: visible until the end of the block "}"
    var task = "Running:"
    fmt.Println(task, done) // => Running: false (this is done from package scope)
    f.Println("Bye bye!")

    // names must be unique only within the same scope
    const done = true // local scoped
    fmt.Printf("done in main(): %v\n", done) // => done in main(): true
    f1()
}
func f1() {
    fmt.Printf("done in f(): %v\n", done) //this is done from package scope
}
```

# Working with arrays

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     // ellipsis operator (dynamic length)
7     a := [...]int{1, 2, 3, 4, 5, 6}
8     a[5] = -6
9     fmt.Println(a)
10    fmt.Printf("%v\n", a)
11    fmt.Printf("%#v\n", a)
12    a[6] = -7
13
14 }
```

PROBLEMS

1

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\arrays> go run .\ellipsis_operator.go
```

```
# command-line-arguments
```

```
.\ellipsis_operator.go:12:3: invalid array index 6 (out of bounds for 6-element array)
```

```
PS E:\_Golang\go_programs\arrays> █
```



```

func main() {
    // In an array literal, the ... notation specifies a length equal to the number of elements in the literal.
    a := [...]int{1, 2, 3, 4, 5, 6}
    a[5] = -6 // this is fine becoz the 5 is valid index for the 6 elements array
    fmt.Println(a)
    fmt.Printf("var a [%d]int", len(a))
    fmt.Printf("a[6] = %d", a[6])
    // a[6] = -100
    // if we wa
    a = append(a, -100)
}

```

```

var a [6]int
invalid argument: a (variable of type [6]int) is not a slice compiler(InvalidAppend)
View Problem No quick fixes available

```

Append is only for Slice not for array. Since array is fixed length even we use the ellipsis operator it has the fixed length. So append won't work.

Just  
array[index]=new\_value

Index must be in range otherwise error

**We cannot add or remove element form the array since they are fixed length.  
We can only modify the existing element**

## Keyed array. Which has no special use

```
16 // keyed array
17 fmt.Println("working with keyed arrays")
18 k := [...]string{
19     | 9: "10th element",
20     | }
21 // here 9 is the index of that value and the values for other indexes
22 // before 9 were zero valued
23 fmt.Printf("k: %#v\n", k)
24 // total 5 elements 3rd index has the value 4 remaining all are zero valued
25 kk := [5]int{
26     | 3: 4,
27     | }
28 fmt.Printf("kk: %#v\n", kk)
29 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\arrays> go run .\ellipsis_operator.go
[1 2 3 4 5 -6]
[1 2 3 4 5 -6]
[6]int{1, 2, 3, 4, 5, -6}
working with keyed arrays
k: [10]string{"", "", "", "", "", "", "", "", "", "", "10th element"}
kk: [5]int{0, 0, 0, 4, 0}
PS E:\_Golang\go_programs\arrays> █
```

```
invalid argument: index 6 (constant of type int) is out of bounds compiler(InvalidIndex)
```

```
View Problem No quick fixes available
```

```
// total 5 elements  
kk := [5]int{3: 4, 6: 7}  
fmt.Printf("kk: %#v\n", kk)
```

Here we can only store upto 5 elements (0 to 4) but i try to add the element for the 6th index which is out of range. So keyed array is not an special one its a usual array but we can see the make that element to that index visibly

```
25     kk := [5]int{
26         3: 4,
27         5,
28         1: 2,
29     }
30     // here the unkeyed value 5 will take the index from the last keyed element.
31     // so 5 will be in the index of 4(last)
32     fmt.Printf("kk: %#v\n", kk)
33 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\arrays> go run .\ellipsis_operator.go
[1 2 3 4 5 -6]
[1 2 3 4 5 -6]
[6]int{1, 2, 3, 4, 5, -6}
working with keyed arrays
k: [10]string{"", "", "", "", "", "", "", "", "", "", "10th element"}
kk: [5]int{0, 2, 0, 4, 5}
PS E:\_Golang\go_programs\arrays> █
```

# Slices

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     s := []int{}
7     fmt.Println(s)
8     fmt.Printf("s: %#v\n", s)
9     println(s == nil)
10
11 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\slices> go run .\working_with_slices.go
[]
s: []int{}
false
PS E:\_Golang\go_programs\slices> █
```

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     s := []int{}
7     fmt.Println(s)
8     fmt.Printf("s: %#v\n", s)
9     println(s == nil)
10    a := s
11    fmt.Println(a == s)
12 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\slices> go run .\working_with_slices.go
```

```
# command-line-arguments
```

```
.\working_with_slices.go:11:16: invalid operation: a == s (slice can only be compared to nil)
```

```
PS E:\_Golang\go_programs\slices> █
```

**We cannot compare two slices**

```
4
5 func main() {
6     s := []int{0, 1, 2, 3, 4, 5}
7     fmt.Println(s)
8     fmt.Printf("s: %#v", s)
9     println(s == nil)
10    a := s[0:3]
11    // fmt.Println(a)
12    res := append(s, a)
13    fmt.Println(res)
14 }
15
```

var a []int  
cannot use a (variable of type []int) as int value in argument to append compiler([IncompatibleAssign](#))  
[View Problem](#) No quick fixes available

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\slices> go run .\working_with_slices.go
# command-line-arguments
.\working_with_slices.go:12:15: cannot use a (type []int) as type int in append
PS E:\_Golang\go_programs\slices> █
```

**We can't append slice to another slice like in python. But if we need to append one slice element one by one we need to use `append` operator**

```
5 func main() {
6     s := []int{0, 1, 2, 3, 4, 5}
7     // fmt.Println(s)
8     // fmt.Printf("s: %#v\n", s)
9     // println(s == nil)
10    a := s[0:3]
11    // fmt.Println(a == s) // invalid operation: a == s (
12    res := append(s, a...)
13    fmt.Println(s)
14    fmt.Println(a)
15    fmt.Println(res)
16 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\slices> go run .\working_with_slices.go
```

```
[0 1 2 3 4 5]
```

```
[0 1 2]
```

```
[0 1 2 3 4 5 0 1 2]
```

```
PS E:\_Golang\go_programs\slices> █
```



## Back ticks in Go strings

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     s1 := "String \"String\""
7     fmt.Println(s1)
8
9     raw_string := `Raw String can be wriiten within "back ticks" \n \t won't wrk`
10    fmt.Println(raw_string)
11 }
12
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\strings> go run .\working_with_strings.go
String "String"
Raw String can be wriiten within "back ticks" \n \t won't wrk
PS E:\_Golang\go_programs\strings> █
```

```
5 func main() {
6     s1 := "String \"String\""
7     fmt.Println(s1)
8
9     raw_string := `Raw String can be wriiten within "back ticks" \n \t won't wrk`
10    fmt.Println(raw_string)
11
12    s := "Hi there Go!"
13
14    fmt.Printf("%s\n", s) // => Hi there Go!
15    fmt.Printf("%q\n", s) // => "Hi there Go!"
16 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\strings> go run .\working_with_strings.go
String "String"
Raw String can be wriiten within "back ticks" \n \t won't wrk
Hi there Go!
"Hi there Go!"
PS E:\_Golang\go_programs\strings> █
```

```
package main
```

```
import "fmt"
```

```
func main() {
```

```
    s1 := "String \"String\""
```

```
    fmt.Println(s1)
```

```
    raw_string := `Raw String can be written within "back ticks" \n \t won't wrk`
```

```
    fmt.Println(raw_string)
```

```
    s := "Hi there  Go!"
```

```
    fmt.Printf("%s\n", s) // => Hi there  Go!
```

```
    fmt.Printf("%q\n", s) // => "Hi there  Go!" - quoted string
```

```
}
```

```
12     s := "Hi there Go!"
13
14     fmt.Printf("%s\n", s) // => Hi there Go!
15     fmt.Printf("%q\n", s) // => "Hi there Go!"
16
17     fmt.Println("The First index has -", s[1])
18     fmt.Printf("The First index has - %c\n", s[1])
19 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\strings> go run .\working_with_strings.go
String "String"
Raw String can be writen within "back ticks" \n \t won't wrk
Hi there Go!
"Hi there Go!"
The First index has - 105
The First index has - i
PS E:\_Golang\go_programs\strings> █
```

```
22
23     str := "țară" // țară means country in Romanian
24     // 't', 'a' , 'r' and 'a' are runes and each rune occupies between 1 and 4 bytes.
25
26     //The len() built-in function returns the no. of bytes not runes or chars.
27     fmt.Println(len(str)) // -> 6, 4 runes in the string but the length is 6
28
29     // returning the number of runes in the string
30     m := utf8.RuneCountInString(str)
31     fmt.Println(m) // => 4
32
33 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\strings> go run .\working_with_strings.go
String "String"
Raw String can be written within "back ticks" \n \t won't wrk
Hi there Go!
"Hi there Go!"
The First index has - 105
The First index has - i
6
4
PS E:\_Golang\go_programs\strings> □
```

```
package main

import "fmt"

func main() {

    // Slicing a string is efficient because it reuses the same backing array
    // Slicing returns bytes not runes

    s1 := "abcdefghijkl"
    fmt.Println(s1[2:5]) // -> cde, bytes from 2 (included) to 5 (excluded)

    s2 := "中文维基是世界上"
    fmt.Println(s2[0:2]) // -> 中 - the unicode representation of bytes from index 0
    and 1.

    // returning a slice of runes
    // 1st step: converting string to rune slice
    rs := []rune(s2)
    fmt.Printf("%T\n", rs) // => []int32

    // 2nd step: slicing the rune slice
    fmt.Println(string(rs[0:3])) // => 中文维
}
```

```
import ("fmt"
        "strings")
func main() {
    // declaring a variable of type func to call the Println function easier.
    p := fmt.Println
    // it returns true whether a substr is within a string
    result := strings.Contains("I love Go Programming!", "love")
    p(result) // -> True
    // it returns true whether any Unicode code points are within our string, and false otherwise.
    result = strings.ContainsAny("success", "xy")
    p(result) // => false
    // it reports whether a rune is within a string.
    result = strings.ContainsRune("golang", 'g')
    p(result) // => true
    // it returns the number of instances of a substring in a string
    n := strings.Count("cheese", "e")
    p(n) // => 3
    // if the substr is an empty string Count() returns 1 + the number of runes in the string
    n = strings.Count("five", "")
    p(n) // => 5 (1 + 4 runes)

// ToUpper() and ToLower() return a new string with all the letters of the original string converted to uppercase or
lowercase.

    p(strings.ToLower("Go Python Java")) // -> go python java
    p(strings.ToUpper("Go Python Java")) // -> GO PYTHON JAVA
    // comparing strings (case matters)
    p("go" == "go") // -> true
    p("Go" == "go") // -> false
```

```
// comparing strings (case doesn't matter) -> it is not efficient
p(strings.ToLower("Go") == strings.ToLower("go")) // -> true

// EqualFold() compares strings (case doesn't matter) -> it's efficient
p(strings.EqualFold("Go", "gO")) // -> true

// it returns a new string consisting of n copies of the string that is passed as the first argument
myStr := strings.Repeat("ab", 10)
p(myStr) // => abababababababababab

// it returns a copy of a string by replacing a substring (old) by another substring (new)
myStr = strings.Replace("192.168.0.1", ".", ":", 2) // it replaces the first 2 occurrences
p(myStr) // -> 192:168:0.1

// if the last argument is -1 it replaces all occurrences of old by new
myStr = strings.Replace("192.168.0.1", ".", ":", -1)
p(myStr) // -> 192:168:0:1

// ReplaceAll() returns a copy of the string s with all non-overlapping instances of old replaced by
new.
myStr = strings.ReplaceAll("192.168.0.1", ".", ":")
p(myStr) // -> 192:168:0:1

// it slices a string into all substrings separated by separator and returns a slice of the substrings
between those separators.
s := strings.Split("a,b,c", ",")
fmt.Printf("%T\n", s) // -> []string
fmt.Printf("strings.Split():%#v\n", s) // => strings.Split():[]string{"a", "b", "c"}
```



```

// If separator is empty Split function splits after each UTF-8 rune literal.
s = strings.Split("Go for Go!", "")
fmt.Printf("strings.Split():%#v\n", s) // -> []string{"G", "o", " ", "f", "o", "r", " ",
", "G", "o", "!"}

// Join() concatenates the elements of a slice of strings to create a single string.
// The separator string is placed between elements in the resulting string.
s = []string{"I", "learn", "Golang"}
j := strings.Join(s, "-")
fmt.Printf("%T\n", j) // -> string
p(j)                  // -> I-learn-Golang

// splitting a string by whitespaces and newlines.
myStr = "Orange Green \n Blue Yellow"
fields := strings.Fields(myStr) // it returns a slice of strings
fmt.Printf("%T\n", fields)      // -> []string
fmt.Printf("%#v\n", fields)     // -> []string{"Orange", "Green", "Blue", "Yellow"}

// TrimSpace() removes leading and trailing whitespaces and tabs.
s1 := strings.TrimSpace("\t Goodbye Windows, Welcome Linux!\n ")
fmt.Printf("%q\n", s1) // "Goodbye Windows, Welcome Linux!"

// To remove other leading and trailing characters, use Trim()
s2 := strings.Trim("...Hello, Gophers!!!?", ".!?")
fmt.Printf("%q\n", s2) // "Hello, Gophers"
}

```

## Structs

```
5  type Person struct {
6      name  string
7      age   int
8      status string
9  }
10
11 func main() {
12     var p1 Person
13     p1.name = "Logesh"
14     p1.age = 23
15     p1.status = "Single"
16
17     fmt.Println(p1)
18
19     var p2 = Person{}
20     fmt.Printf("p2: %#v\n", p2)
21
22     p3 := Person{name: "Someone", age: 3422, status: "Not known"}
23     fmt.Printf("p3: %#v\n", p3)
24 }
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\structs> go run .\working_with_structs.go
{Logesh 23 Single}
p2: main.Person{name:"", age:0, status:""}
p3: main.Person{name:"Someone", age:3422, status:"Not known"}
PS E:\_Golang\go_programs\structs>
PS E:\_Golang\go_programs\structs> █
```

```
package main
import "fmt"
type Person struct {
    name    string
    age     int
    status  string
}
func main() {
    var p1 Person
    p1.name = "Logesh"
    p1.age = 23
    p1.status = "Single"

    fmt.Println(p1)

    var p2 = Person{}
    fmt.Printf("p2: %#v\n", p2)

    p3 := Person{name: "Someone", age: 3422, status: "Not known"}
    fmt.Printf("p3: %#v\n", p3)
}
```

```
5  type Person struct {
6      name  string
7      age   int
8      status string
9  }
10
11 func main() {
12     var p1 Person
13     p1.name = "Logesh"
14     p1.age = 23
15     p1.status = "Single"
16     fmt.Println(p1)
17     fmt.Printf("%v\n", p1)
18     fmt.Printf("%#v\n", p1)
19     fmt.Printf("%+v\n", p1)
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
PS E:\_Golang\go_programs\structs> go run .\working_with_structs.go
{Logesh 23 Single}
{Logesh 23 Single}
main.Person{name:"Logesh", age:23, status:"Single"}
{name:Logesh age:23 status:Single}
PS E:\_Golang\go_programs\structs> █
```

```
// an anonymous struct is a struct with no explicitly defined struct type alias.
diana := struct {
    firstName, lastName string
    age                int
}{
    firstName: "Diana",
    lastName:  "Muller",
    age:      30,
}
fmt.Printf("%#v\n", diana)
// =>struct { firstName string; lastName string; age int }{firstName:"Diana", lastName:"Muller", age:30
/** ANONYMOUS FIELDS **/
// fields type becomes fields name.
type Book struct {
    string
    float64
    bool
}
b1 := Book{"1984 by George Orwell", 10.2, false}
fmt.Printf("%#v\n", b1) // => main.Book{string:"1984 by George Orwell", float64:10.2, bool:false}
fmt.Println(b1.string) // => 1984 by George Orwell
```

```
// mixing anonymous with named fields:
type Employee1 struct {
    name    string
    salary  int
    bool
}

e := Employee1{"John", 40000, false}
fmt.Printf("%#v\n", e) // => main.Employee1{name:"John", salary:40000, bool:false}
e.bool = true         // changing a field
```

# Functions

```
// defining a function that have one parameter of type float64 and returns a value of type float64
func f4(a float64) float64 {
    return math.Pow(a, a)
    //any statements below the return statement are never executed
}

// defining a function that have two parameters of type int and returns two values of type int
func f5(a, b int) (int, int) {
    return a * b, a + b
}

// defining a function that have one parameter of type int and returns a "named parameter"
func sum(a, b int) (s int) {
    fmt.Println("s:", s) // -> s is a variable with the zero value inside the function
    s = a + b

    // it automatically return s
    return // This is known as a "naked" return.
}
```



Interface

```
// declaring an interface type called shape
type shape interface {
    area() float64
    perimeter() float64
}
```

```
// declaring a struct type
type rectangle struct {
    width, height float64
}
```

```
// declaring a struct type
type circle struct {
    radius float64
}
```

Interface is like an abstract method.  
It has only the definition not the function body.

```
// declaring a method for circle type
func (c circle) area() float64 {
    return math.Pi * math.Pow(c.radius, 2)
}
```

```
// declaring a method for circle type
func (c circle) perimeter() float64 {
    return 2 * math.Pi * c.radius
}
```

```
// declaring a method for circle type
func (c circle) volume() float64 {
    return 4 / 3 * math.Pi * math.Pow(c.radius, 3)
}
```

Here the circle type implicitly implements the Shape interface and also the circle has its own methods in addition to the implemented methods.

```
// declaring a method for rectangle type
func (r rectangle) area() float64 {
    return r.height * r.width
}
```

```
// declaring a method for rectangle type
func (r rectangle) perimeter() float64 {
    return 2 * (r.height + r.width)
}
```

Rectangle also implements the shape interface

```
// declaring a function that takes an interface value
func print(s shape) {

    fmt.Printf("Shape: %#v\n", s)
    fmt.Printf("Area: %v\n", s.area())
    fmt.Printf("Perimeter: %v\n", s.perimeter())
}
```

Here comes the advantage of the interface. Now the print function will work for any type that implements the shape interface. So it makes the print function to acts as an polymorphism.

```
func main() {  
  
    // declaring an interface value that holds a circle type value  
    var s shape = circle{radius: 2.5}  
  
    fmt.Printf("%T\n", s) //interface dynamic type is circle  
  
    // no direct access to interface's dynamic values  
    // s.volume() -> error  
  
    // there is access only to the methods that are defined inside the interface  
    fmt.Printf("Circle Area:%v\n", s.area())  
  
    // an interface value hides its dynamic value.  
    // use type assertion to extract and return the dynamic value of the interface  
    value.  
    fmt.Printf("Sphere Volume:%v\n", s.(circle).volume())  
}
```

```
/** TYPE ASSERTIONS**//  
// checking if the assertion succeeded or not  
ball, ok := s.(circle)  
if ok == true {  
    fmt.Printf("Ball Volume:%v\n", ball.volume())  
}
```

This type assertion and type switch only works for Interfaces

```
/** TYPE SWITCHES **//  
  
// it permits several type assertions in series  
switch value := s.(type) {  
case circle:  
    fmt.Printf("%#v has circle type\n", value)  
case rectangle:  
    fmt.Printf("%#v has rectangle type\n", value)  
}
```

## Empty interface

```
// declaring an empty interface value
var empty interface{}

// an empty interface may hold values of any type
// storing an int in the empty interface
empty = 5
fmt.Println(empty) // => 5

// storing a string in the empty interface
empty = "Go"
fmt.Println(empty) // => Go

// storing a slice in the empty interface
empty = []int{2, 34, 4}
fmt.Println(empty) // => [2 34 4]

// fmt.Println(len(empty)) -> error, and it doesn't work!

// retrieving the dynamic value using an assertion
fmt.Println(len(empty.([]int))) // => 3
```



```
// declaring a new struct type which has one field of type empty interface
type person struct {
|   info interface{}
}
}
```

```
    // declaring person value
    you := person{}
```

```
    // assigning any value to empty interface field
    you.info = "You name"
    you.info = 40
    you.info = []float64{4.5, 6., 8.1}
```

```
    fmt.Println(you.info)
```

## wait Group

```
// The pattern to use sync.WaitGroup is:  
// 1. Create a new variables of a `sync.WaitGroup` (wg)  
// 2. Call `wg.Add(n)` where `n` is the number of goroutines to wait for  
// 3. Execute `defer wg.Done()` in each goroutine to indicate to the WaitG  
roup that the goroutine has finished executing  
// 4. Call `wg.Wait()` in main() where we want to block.
```

```
// Declaring two functions: f1 and f2
func f1(wg *sync.WaitGroup) { // wg is passed as a pointer
    fmt.Println("f1(goroutine) execution started" )
    for i := 0; i < 3; i++ {
        fmt.Println("f1, i=", i)
        // sleep for a second to simulate an expensive task.
        time.Sleep(time.Second)

    }
    fmt.Println("f1 execution finished" )

    //3.
    // Before exiting, call wg.Done() in each goroutine
    // to indicate to the WaitGroup that the goroutine has finished executing.
    wg.Done()
    //or:
    // (*wg).Done()
}
```

```
func f2() {
    fmt.Println("f2 execution
started")
    time.Sleep(time.Second)

    for i := 5; i < 8; i++ {
        fmt.Println("f2(), i=", i)

    }
    fmt.Println("f2 execution
finished")
}
```

```
func main() {
    fmt.Println("main execution started")
    // 1. Create a new instance of sync.WaitGroup (we'll call it simply wg)
    // This WaitGroup is used to wait for all the goroutines that have been launched to finish.
    var wg sync.WaitGroup

    // 2.Call wg.Add(n) method before attempting to
    // launch the go routine.
    wg.Add(1) // n which is 1 is the number of goroutines to wait for

    // Launching a goroutine
    go f1(&wg) // it takes in a pointer to sync.WaitGroup

    // No. of running goroutines
    fmt.Println("No. of Goroutines:", runtime.NumGoroutine()) // => 2

    // calling other functions:
    f2()

    // Finally, we call wg.Wait() to block the execution of main() until the goroutines
    // in the WaitGroup have successfully completed.
    wg.Wait()

    fmt.Println("main execution stopped")
}
```

Mutex to avoid the data race. Data race is when many go routines try to access the shared resource

```
const gr = 100
var wg sync.WaitGroup
wg.Add(gr * 2)

// declaring a shared value
var n int = 0

// 1.
// Declaring a mutex. It's available in sync package
var m sync.Mutex
```

```

for i := 0; i < gr; i++ {
    go func() {
        time.Sleep(time.Second / 10)

        // 2.
        // Lock the access to the shared value
        m.Lock()
        n++

        // 3.
        // Unlock the variable after it's incremented
        m.Unlock()

        wg.Done()
    }()
    // Doing the same for the 2nd goroutine
    go func() {
        time.Sleep(time.Second / 10)
        m.Lock()
        defer m.Unlock()
        n--
        wg.Done()
    }()
}
wg.Wait()

// printing the final value of n
fmt.Println(n) // the final final of n will be always 0

```

Mutex to lock and unlock shared resource.  
 To make sure that the resource is accessed by only one go-routines at a time

## Select statement with channels

```
// using select to wait on both goroutines
for i := 0; i < 2; i++ {
    select {
    case msg1 := <-c1:
        fmt.Println("Received", msg1)
    case msg2 := <-c2:
        fmt.Println("Received", msg2)
    }
}
```

The switch case is executed when the case statement receives the value from the channel. That is the the cases is executed , which is received first



main.go X

main.go > generateLineItems

```
3 import (  
4     "math/rand"  
5     "net/http"  
6  
7     "github.com/go-echarts/go-echarts/v2/charts"  
8     "github.com/go-echarts/go-echarts/v2/opts"  
9     "github.com/go-echarts/go-echarts/v2/types"  
10 )  
11  
12 // generate random data for line chart
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

powerl

```
PS E:\_Go_graphs> go build .\main.go  
main.go:7:2: no required module provides package github.com/go-echarts/go-echarts/v2/charts; to add it:  
go get github.com/go-echarts/go-echarts/v2/charts  
main.go:8:2: no required module provides package github.com/go-echarts/go-echarts/v2/opts; to add it:  
go get github.com/go-echarts/go-echarts/v2/opts  
main.go:9:2: no required module provides package github.com/go-echarts/go-echarts/v2/types; to add it:  
go get github.com/go-echarts/go-echarts/v2/types  
PS E:\_Go_graphs> go get github.com/go-echarts/go-echarts/v2/charts  
go: downloading github.com/go-echarts/go-echarts v1.0.0  
go: downloading github.com/go-echarts/go-echarts/v2 v2.2.4  
go get: added github.com/go-echarts/go-echarts/v2 v2.2.4  
PS E:\_Go_graphs> █
```

```
22 func main() {  
23     csv_file := "rcv_test.csv"  
24     readFile, err := os.Open(csv_file)  
25  
26     if err != nil {  
27         fmt.Println(err)  
28     }  
29     fileScanner := bufio.NewScanner(readFile)  
30  
31     fileScanner.Split(bufio.ScanLines)  
32     // reads line by line  
33     for fileScanner.Scan() {  
34         fmt.Println(fileScanner.Text())  
35     }  
36  
37     readFile.Close()  
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS E:\\_Go\_graphs> go run main.go

```
0 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527076082000 2022-03-18 06:55:27.076082 +0000 UTC 0 0  
1 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527097126000 2022-03-18 06:55:27.097126 +0000 UTC 21 0  
2 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527127325000 2022-03-18 06:55:27.127325 +0000 UTC 30 0  
3 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527157525000 2022-03-18 06:55:27.157525 +0000 UTC 30 0  
4 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527187166000 2022-03-18 06:55:27.187166 +0000 UTC 29 0  
5 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527217139000 2022-03-18 06:55:27.217139 +0000 UTC 29 0  
6 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527247354000 2022-03-18 06:55:27.247354 +0000 UTC 30 0  
7 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527277204000 2022-03-18 06:55:27.277204 +0000 UTC 29 0  
8 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527307212000 2022-03-18 06:55:27.307212 +0000 UTC 30 0  
9 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527339258000 2022-03-18 06:55:27.339258 +0000 UTC 32 0  
10 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527367674000 2022-03-18 06:55:27.367674 +0000 UTC 28 0  
11 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527397214000 2022-03-18 06:55:27.397214 +0000 UTC 29 0  
12 00:e0:4c:36:02:8f 00:e0:4c:36:01:29 1647586527427183000 2022-03-18 06:55:27.427183 +0000 UTC 29 0
```



## Go installation

```
logesh@ubuntu:~/Downloads/go_tarfile$ ls
go1.18.linux-amd64.tar.gz
logesh@ubuntu:~/Downloads/go_tarfile$
logesh@ubuntu:~/Downloads/go_tarfile$
```

```
logesh@ubuntu:~/Downloads/go_tarfile$ sudo tar -C /usr/local -xzf go1.18.linux-amd64.tar.gz
[sudo] password for logesh:
logesh@ubuntu:~/Downloads/go_tarfile$ ls /usr/local/
bin  etc  games  go  include  lib  man  sbin  share  src
logesh@ubuntu:~/Downloads/go_tarfile$ ls /usr/local/go
api      bin      CONTRIBUTING.md  doc  LICENSE  PATENTS  README.md  src  VERSION
AUTHORS  codereview.cfg  CONTRIBUTORS     lib  misc     pkg      SECURITY.md  test
```

```
logesh@ubuntu:~/Downloads/go_tarfile$ echo $PATH | grep /usr/local/go/bin
logesh@ubuntu:~/Downloads/go_tarfile$
logesh@ubuntu:~/Downloads/go_tarfile$
logesh@ubuntu:~/Downloads/go_tarfile$ export PATH=$PATH:/usr/local/go/bin
logesh@ubuntu:~/Downloads/go_tarfile$
logesh@ubuntu:~/Downloads/go_tarfile$ echo $PATH | grep /usr/local/go/bin
/home/logesh/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:
/usr/local/games:/snap/bin:/usr/local/go/bin
logesh@ubuntu:~/Downloads/go_tarfile$
logesh@ubuntu:~/Downloads/go_tarfile$
```

```
logesh@ubuntu:~/Downloads/go_tarfile$
logesh@ubuntu:~/Downloads/go_tarfile$ go version
go version go1.18 linux/amd64
logesh@ubuntu:~/Downloads/go_tarfile$
```

```
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ ls
capture1.pcap  get_timestamps_from_packets.go
```

```
Logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ cat get_timestamps_from_packets.go
package main

import (
    "fmt"

    "github.com/google/gopacket"
    "github.com/google/gopacket/pcap"
)

func main() {
    version := pcap.Version()
    fmt.Println("Pcap version:",version)
    fmt.Println("Opening the pcapfile")
    o_handle, _ := pcap.OpenOffline("capture1.pcap")
    defer o_handle.Close()

    packetsource := gopacket.NewPacketSource(
        o_handle,
        o_handle.LinkType(),
    )
    pkt, _ := packetsource.NextPacket()
    fmt.Println("Packet Content:",pkt)
    fmt.Println("Metadata of the Packet :",pkt.Metadata().CaptureInfo)
    aa := pkt.Metadata().CaptureInfo.Timestamp
    fmt.Println("Packet Captured unix timestamp :",aa.Unix())
    fmt.Println("Packet Captured UTC time :",aa.UTC())
    fmt.Println("Packet Captured Local Time :",aa.Local())
}

```

```
Logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ █
```

```
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ go build get_timestamps_from_packets.go
get_timestamps_from_packets.go:6:2: no required module provides package github.com/google/gopacket: go.mod file not found in current directory or any parent directory; see 'go help modules'
get_timestamps_from_packets.go:7:2: no required module provides package github.com/google/gopacket/pcap: go.mod file not found in current directory or any parent directory; see 'go help modules'
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ go mod init working_with_pcaps
go: creating new go.mod: module working_with_pcaps
go: to add module requirements and sums:
    go mod tidy
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
```

```
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ go build get_timestamps_from_packets.go
get_timestamps_from_packets.go:6:2: no required module provides package github.com/google/gopacket; to add it:
    go get github.com/google/gopacket
get_timestamps_from_packets.go:7:2: no required module provides package github.com/google/gopacket/pcap; to add it:
    go get github.com/google/gopacket/pcap
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ go get github.com/google/gopacket
go: downloading github.com/google/gopacket v1.1.19
go: added github.com/google/gopacket v1.1.19
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ go get github.com/google/gopacket/pcap
go: downloading golang.org/x/sys v0.0.0-20190412213103-97732733099d
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
```



New error which means we are progressing

```
Logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ go build get_timestamps_from_packets.go
# github.com/google/gopacket/pcap
../../go/pkg/mod/github.com/google/gopacket@v1.1.19/pcap/pcap_unix.go:34:10: fatal error: pcap.h: No such file or directory
 34 | #include <pcap.h>
    |           ^~~~~~
compilation terminated.
```

<https://github.com/google/gopacket/issues/280>

```
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ sudo apt-get install libpcap-dev
[sudo] password for logesh:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libfprint-2-tod1 libllvm9 libpkcs11-helper1 mobile-broadband-provider-info openvpn
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  libpcap0.8-dev
The following NEW packages will be installed:
  libpcap-dev libpcap0.8-dev
0 upgraded, 2 newly installed, 0 to remove and 291 not upgraded.
Need to get 248 kB of archives.
After this operation, 852 kB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libpcap0.8-dev amd64 1.9.1-3 [244 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu focal/main amd64 libpcap-dev amd64 1.9.1-3 [3,484 B]
Fetched 248 kB in 9s (27.2 kB/s)
Selecting previously unselected package libpcap0.8-dev:amd64.
(Reading database ... 201378 files and directories currently installed.)
Preparing to unpack .../libpcap0.8-dev_1.9.1-3_amd64.deb ...
Unpacking libpcap0.8-dev:amd64 (1.9.1-3) ...
Selecting previously unselected package libpcap-dev:amd64.
Preparing to unpack .../libpcap-dev_1.9.1-3_amd64.deb ...
Unpacking libpcap-dev:amd64 (1.9.1-3) ...
Setting up libpcap0.8-dev:amd64 (1.9.1-3) ...
Setting up libpcap-dev:amd64 (1.9.1-3) ...
Processing triggers for man-db (2.9.1-1) ...
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ go build get_timestamps_from_packets.go
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ ls
capture1.pcap  get_timestamps_from_packets  get_timestamps_from_packets.go  go.mod  go.sum
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
```

```
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ go build get_timestamps_from_packets.go
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$ ./get_timestamps_from_packets
Pcap version: libpcap version 1.9.1 (with TPACKET_V3)
Opening the pcapfile
Packet Content: PACKET: 98 bytes, wire length 98 cap length 98 @ 2021-06-19 23:23:12.214094137 -0700 PDT
- Layer 1 (14 bytes) = Ethernet {Contents[..14..] Payload[..84..] SrcMAC=00:0c:29:6d:ef:1e DstMAC=00:50:56:e0:64:42 EthernetType=IPv4 Length=0}
- Layer 2 (20 bytes) = IPv4 {Contents[..20..] Payload[..64..] Version=4 IHL=5 TOS=0 Length=84 Id=30596 Flags=DF FragOffset=0 TTL=64 Protocol=ICMPv4 Checksum=2027 SrcIP=192.168.234.129 DstIP=8.8.8.8 Options=[] Padding=[]}
- Layer 3 (08 bytes) = ICMPv4 {Contents[..8..] Payload[..56..] TypeCode=EchoRequest Checksum=22182 Id=2 Seq=1}
- Layer 4 (56 bytes) = Payload 56 byte(s)

Metadata of the Packet : {2021-06-19 23:23:12.214094137 -0700 PDT 98 98 0 []}
Packet Captured unix timestamp : 1624170192
Packet Captured UTC time : 2021-06-20 06:23:12.214094137 +0000 UTC
Packet Captured Local Time : 2021-06-19 23:23:12.214094137 -0700 PDT
logesh@ubuntu:~/go_lang_codes/working_with_pcaps$
```

Important note in using the github modules in the go lang

When we import we should import the package not the module.

Ex: github repo name is module under that repo we will be having several folders, these are the packages.

So import should be some thing like this

Import "github.com/user/repo\_name/package\_name"

Steps to get starting to use the github packages in your project.

Go create the folder (your project ) in anypath.

After that under the root folder of your project initialize the go module by

**Go mod init project\_folder\_name**

Now create you files for your project and keep working when you want to use any github packages just import them in the files and start working

Sometimes go will automatically call the **go get -u imported\_pkages** while running the file. Else it will ask as to get so at that time you need to issue the

**Go get github.com/,/..** Command to get the package

Once the mod file is created then in your program file import you packages and the issue the command

**Go build** to get the packages to get download.

# Creating Your Own Go Module

Consider the following steps to create your own Go module and publish it on GitHub.

Create a [free GitHub account](#) if you don't have one.

1. Using the browser, create a new repository on GitHub

e.g. `go_math`

2. Create a directory for the module anywhere on the disk. Inside it create a directory for each package of the module.

The name of the directory will be the name of the module on GitHub.

Write the code for each package.

Don't forget to export all names! That means that the first letter of each name must be uppercase.

3. The next step is to initialize the module by running **`go mod init` and the module path** from the module directory. This will generate **`go.mod`** file that stores the import path and any dependencies.

e.g. **`go mod init github.com/ddadumitrescu/go_math`**

## The executed commands:

```
$mkdir go_math
```

```
$cd go_math/
```

```
$mkdir calc geometry
```

```
$go mod init github.com/ddadumitrescu/go_math eventhough the go_math folder is in different place we need  
init the mod only by the remote repo path so that other can use.
```

```
go: creating new go.mod: module github.com/ddadumitrescu/go_math
```

```
$cat go.mod
```

```
module github.com/ddadumitrescu/go_math
```

```
go 1.13
```

```
$cd ..
```

```
$ls
```

```
go_math
```

*\$tree .*

*.*

*└─ go\_math*

*├─ calc*

*├─ geometry*

*└─ go.mod*

*3 directories, 1 file*



# Publish the Module on GitHub

Move to the module directory (e.g. `cd go_math`)

1. Initialize the local module folder as a git repository.

**git init**

2. Add the remote repository and give it the name origin

**git remote add origin [https://github.com/ddadumitrescu/go\\_math.git](https://github.com/ddadumitrescu/go_math.git)**

Check the name and the URL of the remote repository: **git remote -v**

3. Add all files from the current directory to the staging area.

**git add -A**

4. Set some variables

**git config user.name "andrei"**

**git config user.email "[someone@someplace.com](mailto:someone@someplace.com)"**

5. Commit the changes

```
git commit -m "some init msg"
```

6. Synchronize the local and remote repositories.

```
git push -u -f origin master
```

*Authenticating ...*

Now the local and the remote repositories are synchronized.

**7. Version the module or make the first release.**

Create a git tag:

```
git tag -a v1.0.0 -m "initial release"
```

```
git push origin master --tags
```

*Authenticating ...*



Getting go-eharts package for my codes

```
logesh@ubuntu:~$ cd go_lang_codes/  
logesh@ubuntu:~/go_lang_codes$ ls  
working_with_pcaps  
logesh@ubuntu:~/go_lang_codes$ mkdir working_with_goecharts  
logesh@ubuntu:~/go_lang_codes$ cd working_with_goecharts/  
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$  
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ ls
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ go mod init working_with_goecharts
```

Command 'go' not found, but can be installed with:

```
sudo snap install go          # version 1.17.5, or
sudo apt install golang-go    # version 2:1.13~1ubuntu2
sudo apt install gccgo-go     # version 2:1.13~1ubuntu2
```

See 'snap info go' for additional versions.

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ echo $PATH | grep /usr/local/go
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ export PATH=$PATH:/usr/local/go/bin
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ echo $PATH | grep /usr/local/go
```

```
/home/logesh/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/usr/local/go/bin
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ go mod init working_with_goecharts
```

```
go: creating new go.mod: module working_with_goecharts
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ ls
```

```
go.mod
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ cat go.mod
```

```
module working_with_goecharts
```

```
go 1.18
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ head sample_go.go
package main
```

```
import (  
    "math/rand"  
    "os"  
  
    "github.com/go-echarts/go-echarts/v2/charts"  
    "github.com/go-echarts/go-echarts/v2/opts"  
)
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ cat go.mod
```

```
module working_with_goecharts
```

```
go 1.18
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ go build
sample_go.go:7:2: no required module provides package github.com/go-echarts/go-echarts/v2/charts; to add it:
  go get github.com/go-echarts/go-echarts/v2/charts
sample_go.go:8:2: no required module provides package github.com/go-echarts/go-echarts/v2/opts; to add it:
  go get github.com/go-echarts/go-echarts/v2/opts
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ cat go.mod
module working_with_goecharts

go 1.18
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ go get github.com/go-echarts/go-echarts/v2/charts
go: downloading github.com/go-echarts/go-echarts v1.0.0
go: downloading github.com/go-echarts/go-echarts/v2 v2.2.4
go: added github.com/go-echarts/go-echarts/v2 v2.2.4
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ go get github.com/go-echarts/go-echarts/v2/opts
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ cat go.mod
module working_with_goecharts

go 1.18

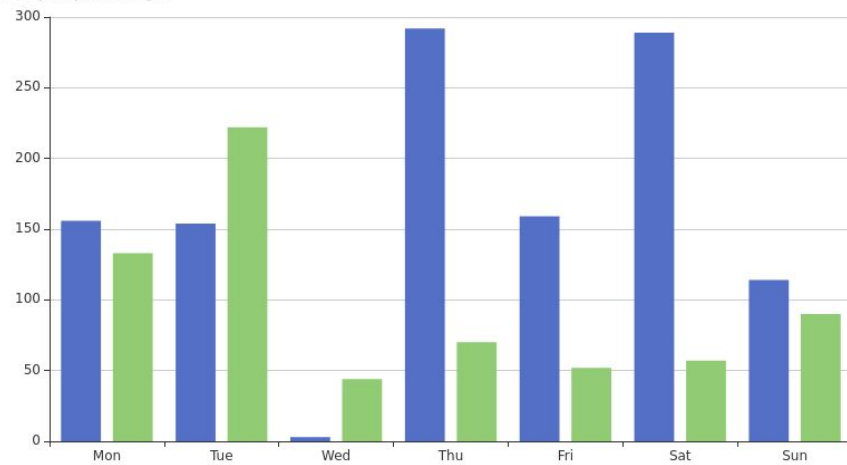
require github.com/go-echarts/go-echarts/v2 v2.2.4 // indirect
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ go build sample_go.go
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ ls
go.mod  go.sum  sample_go  sample_go.go
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
```

```
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ ./sample_go
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$ ls
bar.html  go.mod  go.sum  sample_go  sample_go.go
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
logesh@ubuntu:~/go_lang_codes/working_with_goecharts$
```



## My first bar chart generated by go-echarts

It's extremely easy to use, right?



## Struct type for the json encodings

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 type Person struct {
9     Name string `json:"name"`
10    Age  int64  `json:"age"`
11 }
12
13 func main() {
14     var a Person
15     ee, _ := json.Marshal(a)
16     fmt.Println(string(ee))
17     fmt.Printf("ee: %v\n", ee)
18 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\json> go run .\json_and_types.go
{"name":"","age":0}
ee: [123 34 110 97 109 34 58 34 34 44 34 97 103 101 34 58 48 125]
PS E:\_Golang\go_programs\json> []
```

```
import "encoding/json"

type Person struct {
    Name string `json:"name"`
    Age  int64  `json:"age"`
}

func main() {
    p := Person{
        Name: "John",
        Age:  30,
    }
    b, err := json.Marshal(p)
    if err != nil {
        panic(err)
    }
    fmt.Println(b)
}
```

field Name string  
(main.Person).Name on pkg.go.dev  
struct field tag `json:nam` not compatible with reflect.StructTag.Get: bad syntax for struct tag  
value structtag

[View Problem](#) No quick fixes available

```
Name string `json:"name"`
Age  int64  `json:"age"``
}
```

```
type Person struct {
    Name string `json:"name"`
    Age  int64  `json:"age"``
}
```

Correct format.

```
1 package main
2
3 import (
4     "encoding/json"
5     "fmt"
6 )
7
8 type Person struct {
9     Name string
10    Age int64
11 }
12
13 func main() {
14     var a Person
15     ee, _ := json.Marshal(a)
16     fmt.Println(string(ee))
17     fmt.Printf("ee: %v\n", ee)
18 }
```

It will take the given names for the json encoding if we not given the json string for that struct.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS E:\_Golang\go_programs\json> go run .\json_and_types.go
{"Name":"","Age":0}
ee: [123 34 78 97 109 101 34 58 34 34 44 34 65 103 101 34 58 48 125]
PS E:\_Golang\go_programs\json>
PS E:\_Golang\go_programs\json> █
```

For more about Golang visit

[https://docs.google.com/presentation/d/1einB0LEjUbBpdE0drq\\_9IDe6P\\_rpHM3l\\_gA2XRDVhWI/edit?usp=sharing](https://docs.google.com/presentation/d/1einB0LEjUbBpdE0drq_9IDe6P_rpHM3l_gA2XRDVhWI/edit?usp=sharing)

```
1 package main
2
3 import "fmt"
4
5 type Person struct {
6     Id    int
7     Name string
8     Age  int
9 }
10
11 func main() {
12     var l = Person{Id: 1, Name: "Logesh", Age: 22}
13     fmt.Println(l)
14 }
15
16 func (p *Person) String() string {
17     return fmt.Sprintf("Person Type\nId    : %d\nName : %s\nAge  : %d", p.Id, p.Name, p.Age)
18 }
```



When importing, Go tools looks for the packages inside the `$GOROOT` and `$GOPATH/src` directories. These are simply environment variables and you can set them to any other path as well. But, you don't need to.

`$GOPATH` is called as the *workspace directory* for Go programs. Go source-code, belongs to you and to others lives here. So, when you *import* a package, Go searches that package inside this directory's *src* directory.

