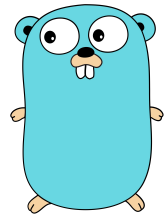
Go from scratch: Beginners-Friendly Guide

by Denis Shchuka

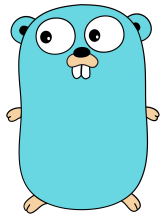
Dive Deeper



Go from scratch: Beginners-Friendly Guide

by Denis Shchuka

Dive Deeper.
Arrays & Slices



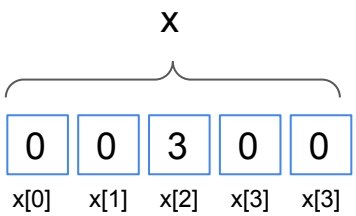
Arrays and Slices

Arrays

```
var x [5]int
```

- ❖ Fixed length
- ❖ Passed by value

```
x[2] = 3
```



Slices

```
var names []string
```

- ★ Dynamically-sized
- ★ Passed by reference

```
names = make([]string, 3)
names[2] = "three"
```

You will use slices much more often than arrays

Every slice has an underlying arras. Basically, slice is reference to a part of underlying array.

Arrays

Example 1. Initialization

```
//array with 2 string elements
var x [2]string
x[0] = "Hello"
x[1] = "world"


//alternative way
x := [...]string{"Hello","world"}
```

Example 2. Access to elements

```
//print first element of the array
fmt.Println(x[0])

//print all array values
for _, v := range x {
    fmt.Println(v)
}
```

Example 3. Copying

```
//copy all values to new array
x2 := x
fmt.Println(x[1]) //"world"
```

Slices

Example 1. Initialization

```
days := []string{“Monday”, “Tuesday”, “Wednesday”, “Thursday”, “Friday”}
```

Example 2. Adding a new element

```
days = append(days, “Saturday”,
“Sunday”)
```

Example 3. Access to elements

```
//Print first 5 elements
fmt.Println(days[:5])

//Print all elements after 5 first
fmt.Println(days[5:])
```

* You can also use for loop to iterate through slice elements

Slices

Example 4. Create reference

```
days2 := days
days2[0] = "X-day"
fmt.Println(days[0]) //"X-day" - now, both slices point out to the same array of elements!!!
```

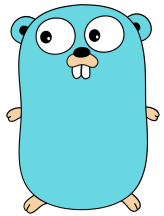
Example 4. Copying

```
days3 := make([]string, len(days))
copy(days3, days)
days3[0] = "Mon"
fmt.Println(days[0]) //"X-day" - days3 points out to new array of elements!!!
```

Go from scratch: Beginners-Friendly Guide

by Denis Shchuka

Dive Deeper.
Structures & Maps.



Structures

Struct is a collection of field that can have different data types.

```
type Employee struct {
    ID       int
    Name     string
    Salary   float64
}
```

- By default, fields of struct instance are zero-valued
- Two struct instances are equal when all their fields are equal
- Structs can be nested. It is a useful way to model more complex structures.

```
var developer Employee
developer.Salary = 5000.0

//other way to declare variable
//of Employee type
boss := Employee{Name: "John", Salary: 20000.0}
```

Maps

Map is one of the most useful data structures

Map is the built-in representation of hash-table data structure in **Go**

Map is set of key-value pairs

Map offers fast lookups, adds, and deletes

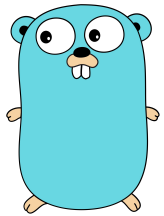
```
map[KeyType]ValueType
```

```
type Coords struct {
     Lat, Long float64
}

var m map[string]Coords
m = make(map[string]Coords)

m["New York"] = Coords {40.730610,
-73.935242}
m["Moscow"] = Coords {55.751244,
37.618423}

fmt.Println(m["Moscow"].Lat)
```

Go from scratch: Beginners-Friendly Guide

by Denis Shchuka

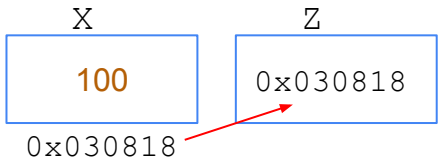
Dive Deeper.
Pointers.



Pointers

Pointer in Go is a variable to store memory address of another variable

```
var X int = 100
Z := &X
```



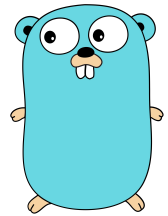

```
//get memory address
fmt.Println(&X)   //0x030818
fmt.Println(Z)    //0x030818
```

```
//get underlying value
fmt.Println(*Z)   //100

//change underlying value via pointer
*Z = 21
fmt.Println(X)  //21
```

Go from scratch: Beginners-Friendly Guide

by Denis Shchuka

Packages & Modules.



Packages & Modules

- Package is one or more .go source code files located in the same directory
- Package contains **logically related** code - types, functions, etc.

The main goal of using packages and modules - to structure you code for better reusability, modularity and maintainability

non-main (non-executable) package (library)

```
package employee

type Employee struct {
      ID    int
      Name string
)
```

main (executable) package

```
package main

import (
      "fmt"
      emp "example.com/employee"
)

func main(){
      var x emp.Employee
}
```

Packages & Modules

- Module is a collection of Go packages in a directory tree

```
<workspace_home>/
     |-- employeepkg
     |-- hrservicepkg
     |-- hrapp
```

To create a module:

1. Create workspace directory tree
2. `go mod init` *`import_path`*
3. Import required packages in main module
4. Redirect Go tools from its module path to the local directory:

```
go mod edit -replace =
example.com/employeepkg =../employeepkg
```

! Every package in module should be located at the same level in the hierarchy of directory tree