BTW: This booklet is in progress. If you would like to participate email me alan.khosro at gmail.

iGo: Golang for idiots

Go is simple by "design". You may say Go is an orthodox religion in programming and software development that denies or rejects most of the recent advancement and conventions. You may also say it is Zen of the software development world. If you think very sophisticated, you are overthinking in Goland. It needs no education but it is not very easy to use either.

Python is easy and convenient (comformist), C++/C#/Java are theoritically advanced (academician), C/Pascal is old-style (orthodox), R/Matlab/SQL are niche players (focused), JS/VBasic/Ruby are supporting lnaguages (UX), Go is Zen for software development (Genuine).

It has a genuis way to implement parallelism, inheritance, safety, and modularity: four destruptive concepts in the last four decades.

To learn Go, you need to think simple like a Zen students. Forget every sophisticated concept like object, class, polymorphism, asynchronism, ...

This booklet aimed programmers with little knowledge in programming and will not use sophisticated concepts. It is an idiot guide written by an idiot programmer for all other idiot programmers.

This booklet encourage you to use the following resourced along with this booklet:

- A Tour Of Go: It is an amazing source to learn basics of Go. I assume you already had a tour of go, if not, do not miss it.
- Go Playground: Whenever you want to test and learn, you can use this playground. We use it throughout this booklet.
- Effective Go: After reading this booklet, whenver you need to learn more, effective Go is an effective source.
- Go Standard Packages: Go's documentation for standard packages is developed to show how Go works in practice. After you advanced a little, use this source as an ultimate learning resource.

Hello World

Let us write our first Go program and print "Hello, World":

```
package main
import "fmt"
func main () {
    fmt.Print("Hello, World.")
}
```

There you go, we printed Hello, World.

We needed standard fmt package to Print. Go, itselg, is very small and most of functions and methods are packaged separately in standard packages that we need to import in our programs.

As you noticed, Go is a structured programming language and modularity in R starts with good (obligatory) practices and smart packaging.
For instance, using main is mandotary. main declares this is the main package (so it is not a package to be used by other packages) and the function main contains the statements to start execution. It is a good practice that Go forces us to follow.

Notice that imported function Print starts with capital letter. The capital letter at the beginning, tells Go the function Print is exported so Go makes it available for other packages that import fmt. So all exported identifiers starts with Capital letter and the rest is for package internal usage and not accessible for others. When writing a package, we need to follow this good practice.

```
import ("fmt"; "time")


func main () {
    fmt.Print("Hello, World\n","你好, 世界\n", "سلام دنیا", "\nNow time is ",
time.Now())
}
```

Data Types

Go is statically-typed language so you can mix integer with string in an statement. Every variable must be declared with its type so that Go knows how to manage its allocated memory and how to handle functions that call that variable.

Go has a variety of boring data types:

```
bool // to store boolean variable true or false
string // to store string: usual texts
int  int8  int16  int32  int64 rune // to store integers
uint uint8 uint16 uint32 uint64 uintptr byte // to store unsigned integer
float32 float64 // to store decimal numbers
complex64 complex128 // to store complex numbers
/*
uint32 means unsigned integer of 32 bits which covers 0 to 2^32-1 integer
numbers.
rune is alias for int32 and represents a Unicode characters. byte is alias for
unit8 and represents a byte.
int, unit, and unitptr are 32 bits in 32-bit operating system and 64 bits in
64-bit operating system.
*/
```

BTW: As you noticed to comment a line //a line is used and to comment a text /* text */ is used.

Variables

Let us write our first program:

In Go syntax, you write as you read so to declare
variable x of type int with initial value 10
you write

```
var x int = 10
```

Interface