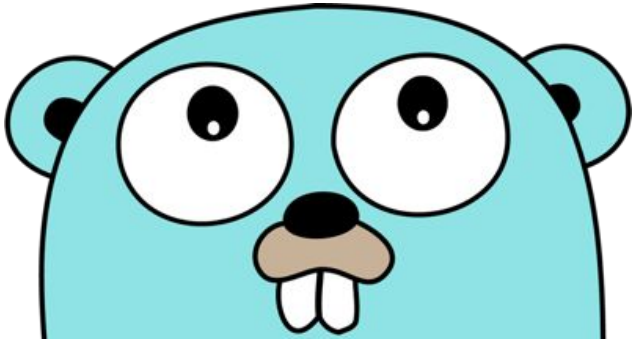

Gopher it!

- A brief introduction to Go Programming Language

Aarjan Baskota
Software Engineer

Why Go was needed

Go came from the need to solve software engineering issues at Google.

As Rob Pike mentions,
“Go’s purpose is not to do research into programming language design; it is to improve the working environment for its designers and their coworkers. Go is more about software engineering than programming language research.”

Robert Griesemer (Hotspot,JVM)

Rob Pike (Unix, UTF-8)

Ken Thompson (B, C, Unix, UTF-8)

Issues

(taken from: https://talks.golang.org/2012/splash.article)

- slow builds—builds would sometime take as long as an hour to complete
- uncontrolled dependencies
- each programmer using a different subset of the language
- poor program understanding (code hard to read, poorly documented, and so on
- duplication of effort
- cost of updates
- version skew
- difficulty of writing automatic tools
- cross-language builds

For Go to succeed, Go must solve these problems

(taken from: https://talks.golang.org/2012/splash.article)

- Go must work at scale for programs with large numbers of dependencies.
- Google needs to get programmers productive quickly in Go, means that the language cannot be too radical.
- Go must be modern.
  It should make efficient use of multi core machines.
  It should have built-in networking and web server libraries.

Top 8 Features

1. Portability
2. Static Type System
3. Performance
4. Rich Standard Library
5. Concurrency and Channels
6. Concrete Types and Interfaces
7. Interfaces
8. Error Handling

1. Portability

Using static linking it actually combines all the dependency libraries and modules into a single binary file based on OS type and arch.

Eg: **GOOS=windows GOARCH=amd64 go build**

List of Environments

2. Static type system

- Declare the types for all your variables
- Define the return type in your functions.
- Indeed, it helps the code more readable and easier to understand.

```
var name string
name = “aarjan”

age := 12       // datatype: int

func Read ( file string ) ( [ ] byte, error ) {
        // expressions
}
```

3. Performance

- Go is statically typed language with automatic garbage collection.
- Each goroutine occupies only 4kb memory space compared to 1Mb for a thread in Java, It is pretty fast even without leveraging its concurrency features.
- No benchmark is perfect, but benchmarking can be a good starting start to know about the capabilities.
  Benchmarking Game Comparing Go vs Java

4. Rich Standard Library

Very rich resources and simple implementation can get most of your work done, without searching for other alternatives.

- net/http : http client/server implementations
- encoding/json : json treated as first class member
- html/template : templating library
- io/ioutil : implements non trivial file and io tasks
- os : represent operating system objects/functionality at a low level that is portable across all go implementations.
-

5. Concurrency and Channels

- First class support.
  Goroutines are Go's lightweight approach to threading and channels are the path to communicate with them.

- With only 4kb memory space occupied by Goroutine, it is possible to have thousands of them running at same time, without much overhead.

- The goroutine and channel based approach to concurrency makes it very easy to use all available CPU cores and handle concurrent IO - all without complicating development.
- Concurrency vs Parallelism - Rob Pike

Binary Tree Problem -> Solution

6. Concrete Types and Methods

- Go has no classes; methods can be added to any type.
- Types stand alone by themselves; they just are and have no hierarchy.
- Methods aren’t special; they’re just functions.
- Methods have a pointer receiver or plain receiver

```
type myFloat float64

func (f MyFloat) Abs() float64{
        If f < 0 {
                return float64(-f)
        }
        return f
}
```

7. Interfaces

- Interfaces enable loosely coupled or decoupled components for your system.

- Just rely on an interface type.
  No need to care who implements the interface or how is it implemented.

- Your controller then can supply a dependency which satisfies the interfaces, i.e. implements all its functions.

- It helps to maintain a clean architecture for unit testing. Now your controller can inject a mock implementation of the interface required by the code, and it would be able to test it it worked or not.

Example on Interfaces, structs and embedding

8. Error Handling

“Error values in Go aren’t special, they are just values like any other, and so you have the entire language at your disposal.” - Rob Pike

- But Go teaches us to check error for each statement.
- It might be quite a redundancy but a lot helpful in debugging.

Error Handling and Go
Handling Errors Gracefully
Handling Repetitive Errors

Go as a OOP

From an Object Oriented standpoint, Go does provides

- the ability to add behavior to your types via methods,
- allows you to implement polymorphic behavior via interfaces
- gives you a way to extend the state and behavior of any existing type via type embedding
- provides a form of encapsulation that allows your types, including their fields and methods, to be visible or invisible.

Strong Ecosystem

Go is not a new language, in fact, it has been 8 years old and matured perfectly to handle enterprise level problems.

Go has great support for many tools we use, from solid libraries for Redis, RabbitMQ, Kafka.

Go's ecosystem is a major win compared to other new languages like Elixir and Rust.

From Docker, Kubernetes, NATS, BoltDB etc. built in Go, many big companies around the world has adopted Go.

https://github.com/golang/go/wiki/GoUsers
https://blog.golang.org/8years

Tooling

Go fmt:
Idiomatic standard matters well in Go community. So, it has built in cl-utility for formatting your code.

Go run:
Compiles and run the code.

Go get:
Downloads library from github and places it in the gopath.

Go doc:
Parses your source code, including comments and produces a documentation in HTML.

Go workspace

A workspace is a directory hierarchy with three directories at its root:

- src contains Go source files,
- pkg contains package objects, and
- bin contains executable commands.

```
.bin/
    hello                          # command executabl
    outyet                          # command executab
pkg/
    linux_amd64/
        github.com/golang/example/
            stringutil.a           # package object
src/
    github.com/golang/example/
        .git/                      # Git repository metad
            hello/
                hello.go           # command source
            outyet/
                main.go            # command source
                main_test.go       # test source
            stringutil/
                reverse.go         # package source
                reverse_test.go    # test source
```

[How to write Go code]

Caveats

- Package Management
  - Dep -> Prototype of dependency management tool for Go
- Lack of Frameworks
  - Idiomatic Go using most out of standard library
  - Some Famous Web-Fr -> Gin, Revel, Iris, Echo etc.
  - DB wrappers -> Squirrel, XO etc.
  - Http Middlewares -> Negroni, Alice etc.
- Inheritance -> Composition with go go using Interfaces and Embedding
- Generics -> some sort of implementation using Empty Interfaces

https://blog.golang.org/toward-go2

Summary

Although, Go was designed to address a set of software engineering issues that Google had been exposed to in the construction of large server software, but in fact the focus on clarity, simplicity and composability throughout the design instead resulted in a productive, fun language that many programmers find expressive and powerful.

- Clear dependencies
- Clear syntax
- Clear semantics
- Composition over inheritance
- Simplicity provided by the programming model (garbage collection, concurrency)
- Easy tooling (the go tool, gofmt, godoc, gofix)

Resources

Official Sources:

- https://tour.golang.org/welcome/1
- http://golang.org/pkg/
- https://github.com/golang/go/wiki
- https://blog.golang.org/index
- Effective Go

Tutorials:

- https://gobyexample.com/
- https://github.com/ardanlabs/gotraining
- https://github.com/avelino/awesome-go

Resources

Articles:

- Go Ten Years and Climbing - Rob Pike
- Naming Conventions - Andrew Gerrand
- Gophercon Slides and videos
- Go-best-practices-2016
- http://gopherdata.io/
- https://medium.com/@benbjohnson/structuring-applications-in-go-3b04be4ff091
- https://goinggo.io/5-not-10-mistakes-i-made-when-first-learning-go-language-576d17906e9a
- https://hackernoon.com/the-beauty-of-go-98057e3f0a7d
- https://medium.com/@divan/how-to-avoid-go-gotchas-744bff599b0c

Resources

Youtube:

- Simplicity is Complicated - Rob Pike
- Just for Func - Francesc Campoy

Slack

- https://gophers.slack.com - Find many channels like data-science, jobs, nepal etc.

Some tips

- A great rule of thumb for Go is the accept interfaces, and return struct types.
- Strings act as a slice too. So, you can perform slicing functions and also range through it using for loop.
- Since keys in a Map cannot be duplicate, you can use it anywhere when you need a set like data structure, or filter the output.
- It’s better to check for error at first and execute 'return' statement.
- Deferred functions are executed in LIFO manner.
- Interfaces are called abstract type and concrete type is used to implement them.
- If you want a singleton object, then make a private instance of the object.
- Empty switch statement can act as if-else chain.

Other Topics

- Defer, Panic and Recover
- Laws of Reflection

Thank You