

GoLang

A Programming Language Case Study

7 November 2018

Kyle Jon Aure
CS Major, WSU - Rochester

Table of Contents

- Introduction
- History
- Support and Uses
- Lexical Elements
- Types
- Composite Types
- Scope
- Testing
- Modularity
- Portability
- Performance

Introduction



Go is a programming language developed by Google Employees.

To help differentiate the word 'go' from the language Go many people say GoLang instead.

People who develop in Go often refer to themselves as Gophers.

Go is mostly in the C family of languages with basic syntax.

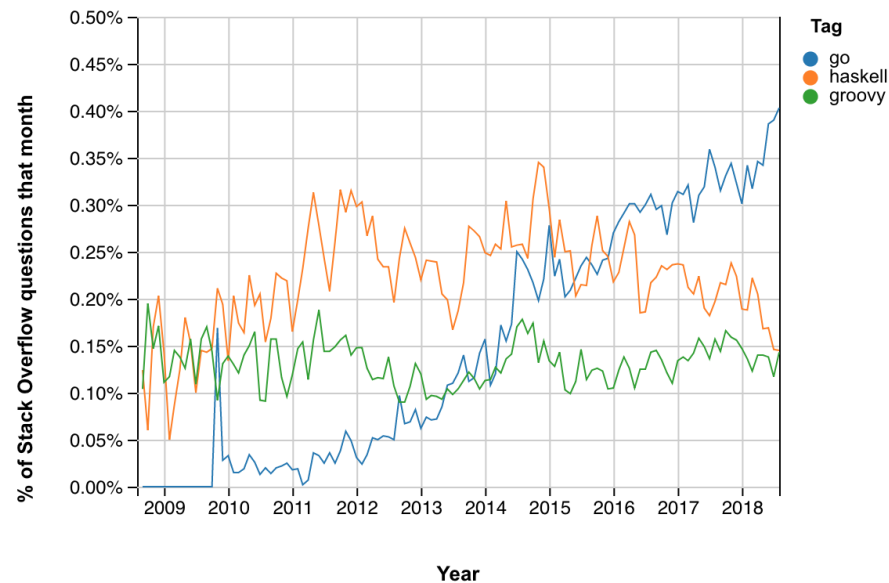
Go was developed as an attempt to combine the ease of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language.

History

Developed by Robert Griesemer, Rob Pike, and Ken Thompson all Google employees.

Started development in 2007 and released November 10, 2009.

Estimated that today there are 504,000 to 966,000 Go programmers.



Trends [Stack Overflow](https://insights.stackoverflow.com/trends?tags=go%2Chaskell%2Cgroovy) (<https://insights.stackoverflow.com/trends?tags=go%2Chaskell%2Cgroovy>)

What is it used for?

Go was developed in response to the boom of cloud computing.

Development of server software using traditional languages like Java and C++ can be complex.

Go is exceptionally good at handling concurrency, garbage collection, and memory management.

Who Uses Go



Support

Development can be done on the Windows, MacOS, and Linux operating systems along with a number of open source operating systems.

Go is compiled into binaries that can be targeted to run on personal computers, servers, and mobile devices.

Go does not have a standalone IDE instead it is supported by existing technology.

- Text Editors: Visual Studio Code, Vim, Emacs, and Atom.
- IDEs: GoLand built on top of IntelliJ, and GoClipse built on top of Eclipse

I tried both Visual Studio Code and Atom and preferred Visual Studio for Go since it allowed me to run terminal commands natively.

Lexical Elements

Numeric Operators:

+ : addition

- : subtraction

* : multiplication

/ : division

% : modulus

Bitwise Operators (int):

& : AND

| : OR

^ : XOR

&^: Clear

<<: Arithmetic shift left

>>: Arithmetic shift right

Language Specification

The Go language specification is written in Backus-Naur Form (BNF).

Identifiers are specified by the production:

identifier = letter { letter | unicode_digit }

Comments are the same as most languages.

- // single line comment
- /* multiline comment */

During compilation white space is ignored.

Except at end of statements new line characters will be replaced with semicolons.

Therefore, semicolons are optional.

Typing

Go is a statically-typed, compiled language.

But, some features of Go make it appear that variables can be dynamically typed.

```
func main() {  
    //tag::typing[]  
    var x int = 5 //Static representation <1>  
    var y = 6      //Dynamic representation <2>  
    z := 7         //Implied representation <3>  
    //end::typing[]  
  
    y = '6'  
  
    //If static typed y == 54 (ascii for '6')  
    //If dynamic typed y == '6'  
  
    fmt.Printf("x is %d, y is %d, and z is %d \n", x, y, z)  
}  
//ENDMAIN
```

[Run](#)

Composite Types

Go supports these eight different composite types; array, struct, pointer, function, interface, slice, map, and channel types.

- Array

```
//tag::array[]  
var a [5]int           //<1>  
a[1] = 5               //<2>  
b := [5]float32{1.2, 1.3, 1.4, 1.5, 1.6} //<3>  
var c [2][3]int        //<4>  
//end::array[]
```

[Run](#)

- Struct

```
//tag::structure[]  
type Dog struct {Name string; Breed string;} //<1>  
d := Dog{Name : "Max", Breed : "Pit Bull"} //<2>  
fmt.Println("My dog's name is ", d.Name)    //<3>  
//end::structure[]
```

[Run](#)

Functions

Functions in Go can take 0 or more arguments, and return 0 or more results.
Functions themselves can be defined as types.

```
//tag::function[]  
func split(sum int) (x, y int) {  
    x = sum * 4 / 9  
    y = sum - x  
    return  
}  
  
type DogFunc func(string, int)  
  
func main() {  
    fmt.Println(split(17))  
  
    DogFuncs := []DogFunc {  
        func(name string, age int){fmt.Printf("Name: %s, Age: %d\n", name, age)},  
        func(name string, age int){fmt.Printf("Age: %d, Name: %s\n", age, name)},  
    }  
  
    DogFuncs[0]("Max", 12)  
    DogFuncs[1]("Toby", 6)  
}  
//end::function[]
```

[Run](#)

Pointers

Pointers work exactly the same as pointers in C++.

(*) symbol is used during function declaration to denote an address pointer.

(&) symbol is used when using a function to pass an address instead of value.

```
//tag::pointer[]  
func print(i *int) {fmt.Println(*i);} //<1>  
  
func main() {  
    j := 5  
    print(&j) //<2>  
}  
//end::pointer[]
```

[Run](#)

Scope

Go uses lexical (static) scoping.

This is similar to languages like Java where scope is expressly defined by the use of curly brackets { }.

```
//tag::scope[]  
var a = 1 //<1>  
  
func main() {  
    fmt.Println("Global scope, a is", a)  
  
    var a = 2 //<2>  
    fmt.Println("Local scope, a is", a)  
    {  
        var a = 3 //<3>  
        fmt.Println("Inner scope, a is", a)  
    }  
}  
  
//end::scope[]
```

[Run](#)

Testing

Go provides natively supported testing using the testing package.

To define a test function the declaration needs to be in the form:

func TestXxx (t *testing.T)

```
//tag::test[]  
  
func add(a int, b int) int { //<1>  
    return a + b  
}  
  
func TestAdd(t *testing.T) { //<2>  
    var a = 5  
    var b = 6  
    var c = add(a, b)  
    if c != 11 {  
        t.Error("Expected ", 11, " got ", c) //<3>  
    }  
}  
  
//end::test[]
```

[Run](#)

Modularity

Similar to Java, Go uses a package / import system to handle large program modularity.

package `main` is reserved for the Go file that contains the main function.

import is an optional declaration to import other local Go files or those that come with the Go language.

Modularity cont.

```
package main

import "errors"
import "fmt"
import "math"

func Sqrt(value float64) (float64, error) {
    if value < 0 {
        return 0, errors.New("Math: negative number passed to Sqrt")
    }
    return math.Sqrt(value), nil
}

func main() {
    result, err := Sqrt(-1)

    if err != nil {
        fmt.Println(err)
    } else {
        fmt.Println(result)
    }
}
```

[Run](#)

Portability

There are three main compilers for Go.

- Go Compiler: written in Go and targets binaries for source machine and across platforms.
- GCCgo: A front end to the open source GCC compiler.
- GopherJS: A Go compiler that targets Javascript

Being a compiled language without an intermediate like Java Bytecode means that target programs are not as portable and typically the source code is needed.

Performance

Go handles concurrence with a technology they call GoRoutines.
This allows for a simple approach to write code that utilizes multiple threads.

```
//tag::goroutine[]  
func say(s string) {  
    for i := 0; i < 5; i++ {  
        time.Sleep(100 * time.Millisecond)  
        fmt.Println(s)  
    }  
}  
  
func main() {  
    go say("world") //<1>  
    say("hello")  
}  
  
//end::goroutine[]
```

[Run](#)

Conclusion

Pros

- Go is very powerful programming language.
- Supports rapid development.
- Focus on Runtime performance.
- Can handle 'big data'.
- Made for cloud development.

Cons

- Lack of operator overloading.
- Type interfaces are not supported.
- Data types are immutable.

Works Cited

Stack Overflow Trends. (2018). Stack Overflow. Retrieved from <https://insights.stackoverflow.com/trends?tags=go%2Chaskell%2Cgroovy>

Cox, R. (2018). How Many Go Developers Are There? Research!rsc. Retrieved from <https://research.swtch.com/gophercount>

Cōngruì, W. (2018). GoLang Minimum Requirements. GoLang. Retrieved from <https://github.com/golang/go/wiki/MinimumRequirements>

Peabody, B. (2018). Server-side I/O Performance. toptal. Retrieved from <https://www.toptal.com/back-end/server-side-io-performance-node-php-java-go>

Pike, R. (2012). Go at Google: Language Design in the Service of Software Engineering. Google. Retrieved from <https://talks.golang.org/2012/splash.article>

Scott, M. L. (2016). *Programming Language Pragmatics*. Morgan Kaufmann Publishing.

Uygur, F. (2018). GoUsers. GoLang. Retrieved from <https://github.com/golang/go/wiki/GoUsers>

FAQ/Origins. (n.a.). GoLang. Retrieved from <https://golang.org/doc/faq#Origins>

Thank you any questions?

Kyle Jon Aure

CS Major, WSU - Rochester

KAure09@winona.edu (mailto:KAure09@winona.edu)

<https://github.com/KyleAure> (https://github.com/KyleAure)

[@Kyle_Aure](http://twitter.com/Kyle_Aure) (http://twitter.com/Kyle_Aure)

Some text

