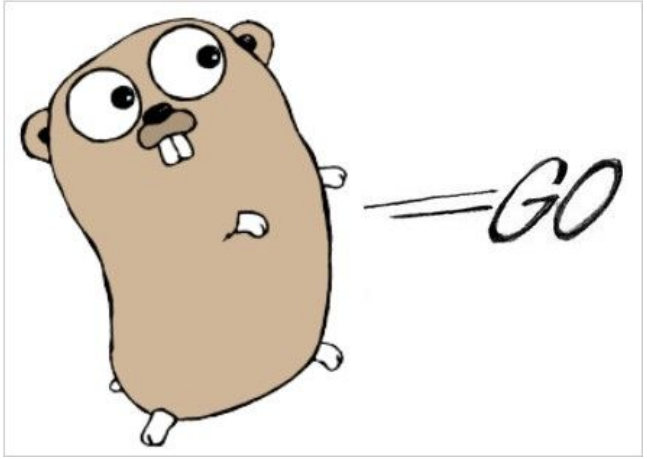
Go
Programming Language

By: Hongzheng Chang, John Ferlic, and Nick Lee

Overview

- Free and open source language
- Created by Robert Griesemer, Rob Pike, and Ken Thompson
- Related to and C
- Used to create standalone executables
  - Fast but poor with UI
    - No GUI toolkit
    - GUI generally integrated into webapp via Chrome
  - Not suited for embedded systems
- Useful for network and web servers
- Partially object-oriented



History

- Project began in 2007 at Google
- Became public open source project in 2009
- Released version 1.0 in 2012
- Follows C syntax
    - Also influenced by Pascal/Modula/Oberon (declarations/packages)

Purpose

Go aims to combine the safety and performance of a statically typed compiled language with the expressiveness and convenience of a dynamically typed interpreted language. It also aims to be suitable for modern systems - large scale - programming.

-Rob Pike OSCON 2010

Design Goals

- Designed to be less syntactically demanding
  - Fewer mandatory keywords
  - Less repetition, less typing
- Does not require IDE
- Scalable
- Supports multi-processors and networking
  - Used for APIs, web servers, web applications (minimal frameworks)
- Intended to be platform-independent

Implementation

- Compiled
  - Fast compilation time
- Statically-typed for primitives
  - Dynamic-typed feel through interfaces
- Parameter passing
  - Everything passed by value
- Emphasis on interfaces over inheritance for efficiency

Memory

- Run-time uses resizable, bounded stacks
  - Goroutines speed up processing by concurrency
    - Independently executing functions (coroutines) run on threads
    - Blocking system call triggers coroutines moved to new, runnable thread
- “New” allocates onto heap
  - Small, confirmed local variables allocated onto stack
- Garbage Collection
  - Mark-and-Sweep
- Memory safety features
  - Assignments type-checked at compile time
  - Type-assertions type-checked at run time

Omissions

- No inheritance
  - Interfaces serve as primary replacement
- No generic types
  - May be added in the future
  - Still looking for clean, simple way to implement
- No classes
  - Only structs and methods
- No pointer arithmetic
- No exceptions
  - Deemed too convoluted
  - Replaced by “defer”, “panic”, and “recover” functions

Syntax

- Line Separation ends statements
  - Semicolon Implicitly placed at the end of the line in the compiler
- Tokens are:
  - Keyword, identifier, constant, string literal, symbol
  - Case sensitive
- Identifier => letter { letter | unicode_digit }
- “=” for assignment, “:=” for joint assignment and declaration
- Uppercase fields: public
- Lowercase fields: private

```
var x int = 2
var x = 2
x := 2
```

```
type die struct {
    numOnDie int
}

func (r die) roll() int {
    rand.Seed( time.Now().UTC().UnixNano())
    rollNum := rand.Intn(r.numOnDie)
    if rollNum == 0 {
        rollNum = 1
    }
    return rollNum
}
```

```
type Animal struct {
    Name string
    mean bool
}
```

Data Types

- Byte
- Numeric
  - Int
  - Float
- Boolean
- Character Strings
- Pointers available for all types
  - “&” to get address
  - “*” to get pointer value

```
var a string = "initial"; //initialize string
var b, c int = 1, 2;      //initialize int
var d = b > c;            //initialize boolean (to false)
```

```
i := 42;  // i assigned to 42
p := &i;  // p set to address of i
*p = 21;  // i set to value 21 through pointer
```

Data Structures

- Arrays - hold same data types

```
var a [10]int
primes := [6]int{2, 3, 5, 7, 11, 13}
```

- Slices - the references to arrays

```
var s []int = primes[0:3]
```

- Structure - User defined data type

```
type struct_variable_type struct {
   member definition;
   member definition;
   ...
   member definition;
}
```

```
type Books struct {
   title string
   author string
   subject string
   book_id int
}
```

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

Data Structure

- Map

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

var m = map[string]Vertex{
    "Bell Labs": {40.68433, -74.39967},
    "Google":    {37.42202, -122.08408},
}

func main() {
    fmt.Println("Map: ", m)
    v1 := m["Google"]
    fmt.Println("Google: ", v1)
    fmt.Println("len:", len(m))
    delete(m, "Google")
    fmt.Println("Map:", m)
}
```

```
Map:  map[Bell Labs:{40.68433 -74.39967} Google:{37.42202 -122.08408}]
Google:  {37.42202 -122.08408}
len: 2
Map: map[Bell Labs:{40.68433 -74.39967}]
```

Real-World Applications

- Used by Google internally
  - Mobile Application
- Also adopted by Netflix, DropBox, SoundCloud
  - Server friendly
- Docker: large-scale software-containerization
  - Static compilation
  - Extensive standard library and data types
  - Full development environment
    - go fmt (solve “tabs vs. spaces” once for all)
    - go test (runs all Test* functions in *_test.go)
- Server architecture and SQL (structured query language)
  - Database installation
  - Database packages

More information

- https://golang.org/doc/faq#history
- https://gobyexample.com/maps
- https://www.docker.com/docker-community
- https://www.goinggo.net/2013/07/object-oriented-programming-in-go.html
- https://www.golang-book.com/books/intro

Demo

```go
package main

import (
  "time"
    "bufio"
    "fmt"
    "math/rand"
    "os"
)

func main(){
  reader := bufio.NewReader(os.Stdin)
  fmt.Print("Enter Question: ")
  text, _ := reader.ReadString('\n')
  fmt.Println("Asking magic 8 Ball: ")
  fmt.Println(text)
  var magicBall int ;

  s1 := rand.NewSource(time.Now().UnixNano())
  r1 := rand.New(s1)
  magicBall = r1.Intn(7);
  switch magicBall {
  case 0:
        fmt.Println("Maybe");
    case 1:
        fmt.Println("Probably not");
    case 2:
        fmt.Println("Ask again later");
    case 3:
        fmt.Println("Seems likely");
    case 4:
        fmt.Println("A distinct possibility");
    case 5:
        fmt.Println("Not a chance");
    case 6:
        fmt.Println("Hard to say");
      }
}
```