The Go Programming Language

Alan Dewar

2023-03-28

The Go Programming Language

- Who’s Using Go
- History and Philosophy
- Key Features and Omissions
- Language Constructs
- Error Handling
- Concurrency
- Standard Library
- Tools
- Quirks
- Learning Go

My Background

- MSc, Computer Science
- CUUG Board of Directors since 1998
- Synopsys since 2017

Languages I’ve Used

- APL
- Assembly languages
  - COMPASS (CDC)
  - ALM (Multics)
  - 6502
  - 68020
  - SPARC
- BASIC
- C, C++
- FOCAL
- FORTRAN, Fortran
- Go
- Java
- JavaScript
- Lisp, Scheme
- PL/I
- Pascal
- PostScript
- Prolog
- Ruby
- Shell: sh, bash, csh
- Simula
- SNOBOL
- Tcl

Who's Using Go

- Google
  - Core data solutions team: web indexing services
  - Chrome content optimization service
  - Firebase hosting team: static web hosting services
  - Site reliability engineering team
- Docker, Kubernetes
  - Go is the language of containers, cloud

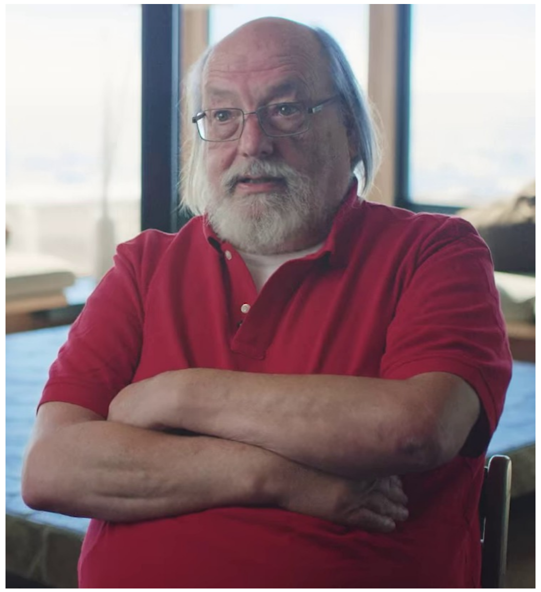
Who’s Using Go

- PayPal
- Netflix
- Meta
- Uber
- Dropbox
- Ethereum
- GitLab
- Synopsys

History

Initial Whiteboard Session, 2007-09-21

- Robert Griesemer: Google
- Rob Pike: Unix, Plan 9, UTF8
- Ken Thompson: Unix, Plan 9, UTF8



Rob Pike photo by Kevin Shockey - https://www.flickr.com/photos/shockeyk/4833152910/in/photostream/, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=33256884

History

Additional Key Team Members, 2008

- Ian Lance Taylor: gcc, Taylor UUCP, gccgo
- Russ Cox: Google, interfaces, I/O library

History

Releases

- 2009-11-10: Public open source project
- 2012-03-28: go1
- 2023-02-01: go1.20
- Every six months

History

Designed by Google to help solve Google’s problems

- Slow builds
- Uncontrolled dependencies
  - C/C++: #include, #ifndef _SYS_STAT_H
- Each programmer using a different subset of the language
- Code hard to read, poorly documented
- Cost of updates
- Difficulty of writing automatic tools

Philosophy

- Influenced by Plan 9 from Bell Labs
- Focus on clarity, simplicity
  - Clean syntax, few keywords
  - Simple grammar
- All three:
  - Efficient compilation
  - Efficient execution
  - Ease of programming
- Familiar
  - C-like

Philosophy

Compatibility

- Go 1 compatibility promise
  - Programs should continue to work through all 1.x releases
- Exceptions
  - Security
  - Undefined behavior
  - Spec errors
  - Bugs
  - New fields, methods, and exports
  - Package “unsafe”

Key Features

- Fast compilation to native binary
- Static typing
- Automatic garbage collection
- Static initialization
- Error handling
- Concurrency
- Reflection
- Modern standard library
- Tools

Key Omissions

- Header files (#include)
- Forward declarations
- Inheritance
- Unions
- Pointer arithmetic
- Implicit type conversion
    - Including numeric
- Default function arguments
- Function name overloading
- Type aliases
- Assignment as expression
- ++, -- as expression
- “implements”
- Exceptions
- Ternary ?: operator
- Assertions

Notable Fatal Errors

- Cyclic dependencies
- Unused variables
- Unused imports

Hello World

```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, 世界")
}
```

Language Constructs: Types

- bool
- int, int8, int16, int32, int64, uint, uint8, uint16, uint32, uint64
- float32, float64
- complex64, complex128
- Array, slice
- struct
- Pointer
- Function
- interface
- map
- channel

Language Constructs: Declarations

- Declarations
  - Go
    ```
    var fn func([]int) int
    type T struct { a, b int }
    ```
  - C
    ```
    int (*fn)(int[]);
    struct T { int a, b; }
    ```
- Scope
  - Exported (visible to importers): start with capital
  - Unexported (not visible to importers): start with lower-case

Language Constructs: Statements

-
  ```
  foo := bar
  a, b = b, a
  ```

-
  ```
  for i := 0; i < 10; i++ { … }
  for i, thing := range things { … }
  ```

-
  ```
  if x < 0 { … } else { … }
  ```

-
  ```
  switch runtime.GOOS {
  case "darwin":
      fmt.Println("OS X.")
  case "linux":
      fmt.Println("Linux.")
  default:
      fmt.Println("Other.")
  }
  ```

-
  ```
  func main() {
      defer fmt.Println("world")
      fmt.Println("hello")
  }
  ```

Language Constructs: Methods

- func with receiver

```
type Vertex struct { X, Y float64 }
func (v Vertex) Abs() float64 {
    return math.Sqrt(v.X*v.X + v.Y*v.Y)
}
```

- Not limited to structs

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

Language Constructs: Interfaces

- Set of methods
- Duck typing

```
type Abser interface {
    Abs() float64
}

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

func main() {
    var a Abser = MyFloat64(-math.Sqrt2)
    fmt.Println(a.Abs())
}
```

Error Handling

- error type

```
type error interface {
    func Error() string
}
```

- Function return convention

```
func foo() (string, error) {
    if happy() {
        return "Good!", nil
    } else {
        return "", fmt.Errorf("I am not happy.")
    }
}
```

Concurrency

- goroutines
  - Light-weight threads
  - Shared memory
  - `go f(x, y, z)`
- Channels
  - Sending and receiving values of a given type
  - Receiver blocks until a value is available
  - Multiple senders and/or receivers
- Don’t communicate by sharing memory; share memory by communicating.

Concurrency: Example

```
func fibo(c, quit chan int) {
   x, y := 0, 1
   for {
      select {
      case c <- x:
         x, y = y, x+y
      case <-quit:
         fmt.Println("quit")
         return
      }
   }
}
```

```
func main() {
   c := make(chan int)
   quit := make(chan int)
   go func() {
      for i := 0; i < 10; i++ {
         fmt.Println(<-c)
      }
      quit <- 0
   }()
   fibo(c, quit)
}
```

Standard Library (some examples)

- archive: tar, zip
- compress: bzip2, gzip, lzw, …
- crypto: aes, des, ecdsa, md5, rand, rsa, sha512, tls, x509, …
- database: sql
- encoding: base64, json, xml, …
- fmt
- html
- image: gif, jpeg, png, …
- io
- math
- net: http, mail, url, …
- os
- reflect
- regexp
- sync
- testing
- time

Tools

- go run
- go build
- go mod
- go test
- go vet
- gofmt
- godoc
- cgo

Quirks

- Function’s open brace must be on same line as “func”
- Implicit semicolon
- Constants are untyped
- Interface holding nil value is not equal to nil
- goroutine scope

```
for _, v := range values {
    v := v    // This is essential!
    go fmt.Println(v)
}
```

- Date format

```
fmt.Println(time.Now().Format("2006-01-02 15:04:05 -0700"))
```

Learning Go

- Official web site: https://go.dev/
- A Tour of Go: https://go.dev/tour/
- The Go Playground: https://go.dev/play/
- Language specification: https://go.dev/ref/spec
- Standard packages: https://pkg.go.dev/std
- Effective Go: https://go.dev/doc/effective_go

A Tour of Go

Questions?

Thank You