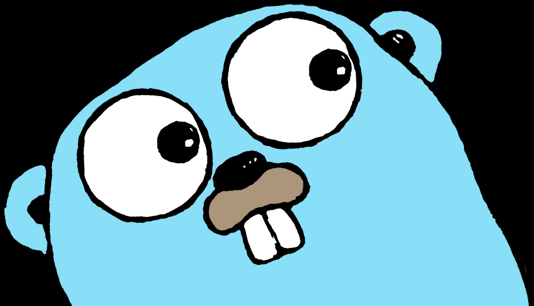
The Go Programming Language

Frank Roberts
frank.roberts@uky.edu



- C++ (1983), Java (1995), Python (1991): not modern
- Java is 18 years old; how has computing changed in 10?
    - multi/many core
    - web programming is everywhere
    - massive parallel and distributed systems
- These languages are not designed for today's environment
- Google designed Go to deal with shortcomings of current systems-level languages
- Go is designed to make writing code on modern systems easier and more natural.

- What makes Go modern?
  - Maps and slices are built in.
  - Garbage collection is built in.
  - Concurrency is built in.

- What makes Go better?
  - Good design choices simplify the language.
  - A new approach to encapsulation
  - A better concurrency model

```go
package main

import "fmt"

func main() {

  fmt.Print("Hello, World.\n")

}
```

- Slices and Maps are built in flexible structures.
- Slices
    - More flexible than arrays
    - Similar to lists in Python
    - Support for slicing operations: myslice[start:end]

```go
func main() {
  fib := []int{0, 1, 1, 2, 3, 5, 8, 13}
  fmt.Println(fib[3])
  fmt.Println(fib[5:7])
  fmt.Println(fib[:3])
  fib = append(fib, 21)
  fmt.Println(fib[3:])
}
```

Output:

```
2
[5 8]
[0 1 1]
[2 3 5 8 13 21]
```

- Maps
    - Associate keys with values
    - Keys may be almost any type (== must be defined)
    - simple literal syntax
    - fetch of non-existent key results in zero value
- Compose slices and maps for simple data structures

```go
func main() {
  attended := map[string] bool{
                "Ann": true,
                "Joe": true}
  fmt.Println(attended["Ann"])
  fmt.Println(attended["Bill"])
  present, ok := attended["Paul"]
  fmt.Println(present, ok)
}
```

Output:

```
true
false
false false
```

- Concurrency model: “Share memory by communicating”
- Goroutines
    - More lightweight than threads
    - Say “go foo()” to run foo concurrently
    - Similar to backgrounding in a Linux shell with '&'
- Channels
    - Like Unix pipes
        - channels are typed
        - Programmer has full control over buffering
    - May be of any type, including channels
- Structure concurrency so that synchronization is implicit in the communication patterns.

- Example: Testing to find prime numbers
- Use a manager-worker model
  - Manager spawns a number of testing routines
  - Each routine tests a different portion of the range
  - Testers send primes to manager over a single channel
  - Testers send a flag value over channel before exiting
- Manager collects primes as they are comupted
- Manager sorts and prints list

The testing routine:

```go
package main

func test_range(start, stop, step int, res chan int) {

  for i := start; i < stop; i += step {
    prime := true
    if i % 2 == 0 && i != 2 { prime = false }
    for j := 3; j*j <= i && prime; j += 1 {
      if i % j == 0 {
        prime = false
      }
    }
    if prime {res <- i}
  }
  res <- 0
}
```

Spawn goroutines:

```
  runtime.GOMAXPROCS(NCPU)

  res := make(chan int, buf)
  for i := 0; i < NCPU; i++ {
    go test_range(i+1, end, NCPU, res)
  }
```

Collect prime numbers into a slice:

```
alldone := 0
for alldone < NCPU {
 next = <-res
 if next != 0 {
  primes = append(primes, next)
 } else {
  alldone += 1
 }
}
```

- -Reading and constructing types
  - - Reads left to right always

English declaration	**C declaration**	**Go declaration**

-Reading and constructing types

- Reads left to right always

English declaration	C declaration	Go declaration
declare foo as array 10 of int	`int foo[10]`	`var foo [10]int`

-Reading and constructing types

- Reads left to right always

English declaration	C declaration	Go declaration
declare foo as array 10 of int	`int foo[10]`	`var foo [10]int`
declare foo as array of pointer to int	`int *foo[]`	`var foo []*int`

- Reading and constructing types
    - Reads left to right always

English declaration	**C declaration**	**Go declaration**
declare foo as array 10 of int	`int foo[10]`	`var foo [10]int`
declare foo as array of pointer to int	`int *foo[]`	`var foo []*int`
declare foo as array of pointer to function returning int	`int (*foo[])()`	`var foo []func () int`

-Reading and constructing types

- Reads left to right always

English declaration	C declaration	Go declaration
declare foo as array 10 of int	`int foo[10]`	`var foo [10]int`
declare foo as array of pointer to int	`int *foo[]`	`var foo []*int`
declare foo as array of pointer to function returning int	`int (*foo[])()`	`var foo []func () int`

```
int (*(*foo)(int (*)(int , int ), int ))(int , int )
```

```
var foo func(func(int, int) int, int) func(int, int) int
```

declare foo as pointer to function (pointer to function (int, int) returning int, int) returning pointer to function (int, int) returning int

- Dependency Analysis

- Poor dependency analysis hurts compile time
  - C-style includes are difficult to analyze at compile time
  - include guards don't prevent extra reads
  - Example: KOAP my own 1200 line C++ project
    - includes 129 headers 837 times total
    - top-level C++ file includes 122 headers 149 times
  - Example: Google binary (instrumented in 2007)
    - Opens hundreds of headers tens of thousands of times
    - 4.2MB of source expands to 8GB
    - Builds take approximately half an hour on a distributed build system

- Go defines dependencies as part of the language
- The dependencies of a Go package are always computable
  - Circular dependencies are not permitted
  - imports for unused packages are compilation errors
  - Go's dependency model isn't new
- The Go compiler spends less time reading dependencies
  - No more than one file read per import
  - Export info goes at the top of a compiled package
- Google instrumented the build of large Go program
  - Code fanout is 50x better than the C++ example
  - Builds take seconds, not minutes

- What I didn't mention
    - Go takes a new and better approach to encapsulation
    - Go has:
        - First class function values
        - A large standard library
        - A tool for building, analyzing, testing, documenting, formating, and fixing code
    - Even more little things...
- Why use Go?
    - Modern features in a compiled language
    - Go is fun to write

References and Resources:

- Effective Go: http://golang.org/doc/effective_go.html
- The Go Programming Language: http://golang.org
- Go at Google: Language Design in the Service of Software Engineering: http://talks.golang.org/2012/splash.article
- The Go Programming Language Specification: http://golang.org/ref/spec
- Go Playground: http://play.golang.org
- A Tour of Go: http://tour.golang.org

Me: www.jafrro.net, frank@jafrro.net