St Louis Java Users Group

12 April, 2018

Charles A Sharp
Principal Software Engineer
csharp@objectcomputing.com
slides: http://speakerdeck/csharp

OCI | WE ARE SOFTWARE ENGINEERS.

Disclaimer

I can't talk about Go

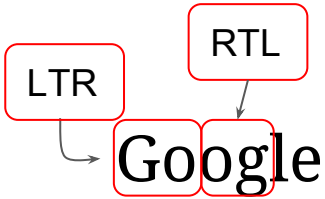
Go {around,down,in,into,on,out,up}

Goes without saying

Go for it

Disclaimer

Go begging



I can't talk about Go

Go there

Don't Go there

Go for the gold

Go long

Go crazy

Go to the mat(tresses)

Go one better

Go get it

Go fly a kite

Go away

Go for broke

Get goin'

I could go on...

Tonight's Best Laid Plans:

- What is this Go thing?
- Go's origins: the people and their languages.
- Language aims and purposes.
- Reading Go:
  - Some are the same
  - Some seem the same but are different
  - Some are just flat different

Fibonacci in Go!

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}


func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
		c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

```go
package main

import "fmt"

func main() {
	var iterations int = 10

	c := fibonacci()

	for n := 0; n < iterations; n++ {
		fmt.Printf("%d ", <- c)
	}
	fmt.Println()
}

func fibonacci() chan int {
	c := make(chan int)
	go func() {
		for i, j := 0, 1; ; i, j = i+j, i {
			c <- i
		}
	}()
	return c
}
```

The Go Programming Language

Go is an open source programming language that makes it easy to build **simple**, reliable, and efficient software.

https://go.googlesource.com/go

…Language Goals...

Evolution of Go - Robert Griesemer (Gophercon 2015)
https://www.youtube.com/watch?v=0ReKdcpNyQg

The task of the programming language designer "is consolidation, not innovation."

("Hints on programming-language design", Hoare, 1973)

…Language Goals

Guiding principles:

- **Simplicity**, safety, and readability are paramount.
- Striving for orthogonality in design.
- Minimal: One way to write a piece of code.
- It's about **expressing algorithms**, not the type system.
- *Collective unconscious history of programming languages.*

Who designed Go?

- Ken Thompson
- Rob Pike
- Robert Griesemer

Ken Thompson...

- Regular Expressions - QED (Multics/BCPL) (mid-60s)
- Unix(69/73), Plan 9(mid-80s/92), Inferno(90's/96)
- ACM Turing Award - 1983 ("Reflections on Trusting Trust")
- Co-creator UTF-8 (1992)
- Bell Labs: 1966 - 2000

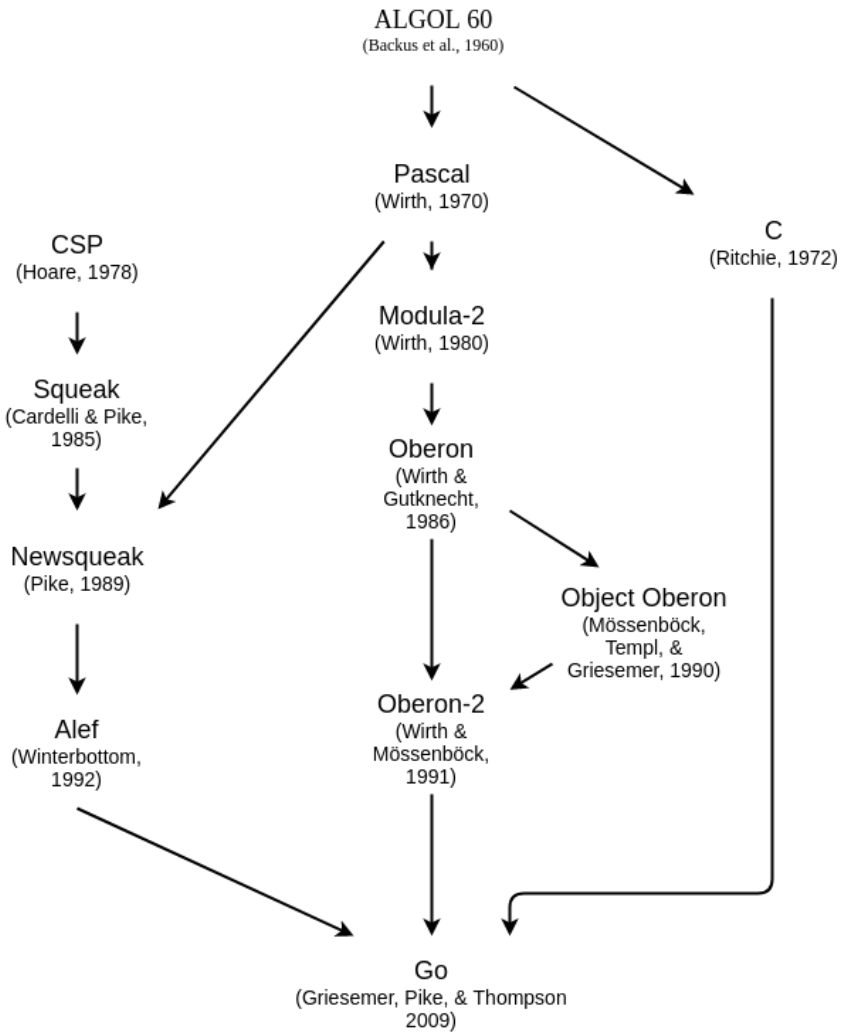
Rob Pike

- Wrote the first window system for Unix in 1981.
- Squeak (w/Cardelli) and NewSqueak (1985)
- Co-creator UTF-8 (1992)
- Worked on Sawzall
- Co-author (w/Brian Kernighan):
  - The Unix Programming Environment (1984)
  - Practice of Programming (1999)
- Bell Labs: 1980 - 2002

Robert Griesemer

- PhD ETH Zurich, Switzerland 1993
- Studied and worked Nicholas Wirth Institute
- Strongtalk team at Longview (bought by Sun 1997)
- Java Hotspot
- Code generation for Google's V8 JavaScript engine
- Design/Implementation of Sawzall
- Did ***not*** work at Bell Labs

History… Predecessors: (Not just C)



Russ Cox (rsc)

Interview: http://www.pl-enthusiast.net/2015/03/25/interview-with-gos-russ-cox-and-sameer-ajmani/

I was finishing my PhD in spring 2008 and visited Google. I had worked with Rob at Bell Labs, and both he and Ken told me about Go, and I was hooked. When I joined the team in August, the language was still just a prototype, with almost no library. I took over the compiler and runtime, and I got to help to develop the standard library and all the revisions and refinements to the language prompted by that experience. Today, Rob and I lead the overall Go project at Google together.

When all is (being) said and (being) done.

- simple, minimalistic and consistent language design
- open source
- C-style syntax
- statically typed variables
- managed memory using garbage collection
- compiled into stand-alone executables
- concurrency features built into the language core
- comprehensive standard library
- fast compilation and execution speed
- tooling addresses traditional problems, i.e., formatting and documentation

What's is it good for?

… especially well-suited for programs that involve networking or other concurrent tasks; goroutines are very convenient and efficient, and there is also good support for more traditional shared-memory approaches. Empirically, people who write new networking code tend to like Go. I personally would use it for anything where in the past I might have used C or Java or C++.

Brian Kernighan
https://features.slashdot.org/story/15/11/18/1748247/interviews-alan-donovan-and-brian-kernighan-answer-your-questions

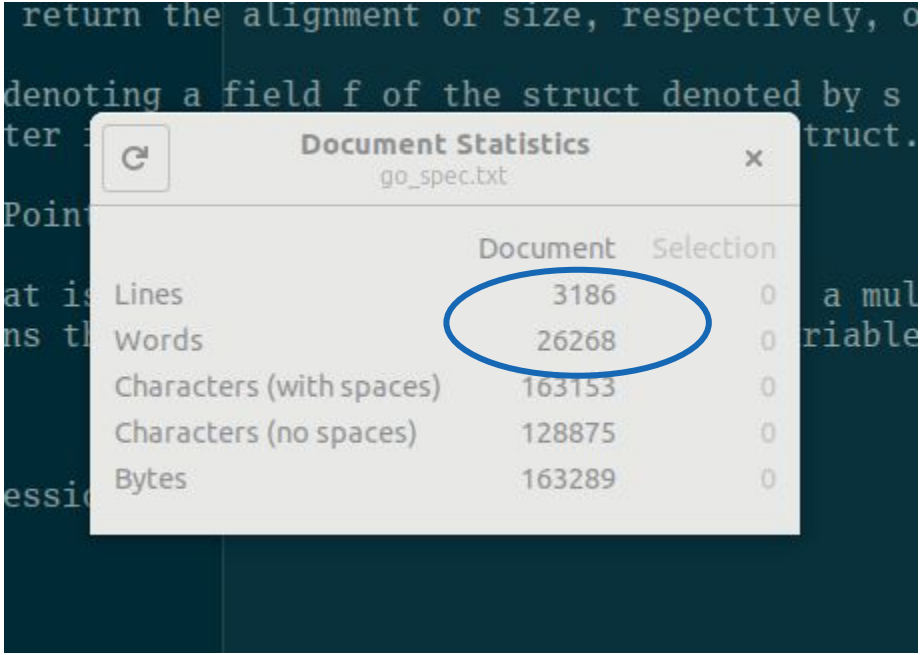
Go Open Source: 10 November, 2009

Go 1.0: March 28, 2012

Go 1.10: 16 February, 2018

26268 / 300 = 87.56
26268 / 450 = 58.37

Go Spec, 1.10, 16 Feb, 2018



Resources

The Go Programming Language: https://golang.org/

Documentation Page: https://golang.org/doc

- A Tour of Go
- How To Write Go
- Effective Go
- FAQ
- Specification

Go Blog: https://blog.golang.org/

Go Talks: https://talks.golang.org/

Let's Go play!

https://golang.org

https://play.golang.org

Language...

Types

Keywords

Function signatures

Packages

Initialization and execution

Tools

Comments

// -- line

/* … block ... */

// Comments right before package are package comments.

Comments do ***not*** nest

Comments are used by go doc for automated document generation.

go doc versus godoc

Printf verbs

- %x %o %b — hex, octal, binary
- %d — decimal integer
- %f %g %e — float formats
- %t — boolean true or false
- %c %s => %q — rune, string, quoted as appropriate
- %v — fitting format
- %T — type
- %% — %

increment/decrement are statements

- Only postfix; no prefix
- Stands alone (pretty much)
- As statements -- not expressions -- can not do:

```
j = i++

a = b[idx++]
```

Keywords

```
break    default     func      interface   select
case     defer       go        map         struct
chan     else        goto      package     switch
const    fallthrough if        range       type
continue for         import    return      var
```

const

```
const (
    i = 1
    j = 2
    k = 3
)
```

vs

```
const (
    i = 0
    j
    k
)
```

vs

```
const (
    i = iota
    j
    k
)
```

var

A variable declaration creates one or more variables, binds corresponding identifiers to them, and gives each a type and an initial value.

```
var i int
var U, V, W float64
var k = 0
var x, y float32 = -1, -2
var (
    i       int
    u, v, s = 2.0, 3.0, "bar"
)
var re, im = complexSqrt(-1)
var _, found = entries[name]
```

Variables … Four ways to create a variable

```
1.  var stringVarName string = "JUG"
2.  var stringVarName = "JUG"
3.  var stringVarName string   // initialized to ""
4.  stringVarName := "JUG"
```

Blank Identifier

_ (Underscore)

```
for _, arg := range os.Args[1:] {
    fmt.Println(arg)
}
```

Types

Boolean

Numeric

String

Array

Slice

Struct

Pointer

Function

Interface

Map

Channel

Numeric types

- uint8, uint16, uint32, uint64
- int8, int16, int32, int64
- float32, float64
- complex64, complex128
- byte alias for uint8
- rune alias for int32

Language - Predefined Numeric types

There is also a set of predeclared numeric types with implementation-specific sizes:

- uint — either 32 or 64 bits
- int — same size as uint
- uintptr — an unsigned integer large enough to store the uninterpreted bits of a pointer value

rune

A rune literal represents a rune constant, an integer value identifying a Unicode code point. A rune literal is expressed as one or more characters enclosed in single quotes, as in 'x' or '\n'. Within the quotes, any character may appear except newline and unescaped single quote. A single quoted character represents the Unicode value of the character itself, while multi-character sequences beginning with a backslash encode values in various formats.

https://golang.org/ref/spec#Rune_literals

type

A type determines a set of values together with operations and methods specific to those values.

```
type (
    // Point and struct{...} are different types
    Point struct { x, y float64 }

    // polar and Point denote different types
    polar Point
)
```

Data structures...

- Arrays
  - numbered sequence of elements
  - fixed number of elements
  - number of elements is the length; never negative; found with len, e.g., `length = myAry.len()`
- Slices
  - contiguous segment of an underlying array.
  - slice created with make always allocates a new, hidden array.
  - len is discovered with len(), capacity is discovered with cap()
  - number of elements can be extended with append().

...Data structures...

- Maps
  - unordered set of elements on one type
  - indexed key-value pairs
  - created with make
- Structs
  - sequence of named elements (fields) with a name and type
  - names may be explicit or implicit
  - within a struct, names must be unique

Slice

A slice is a descriptor for a contiguous segment of an *underlying array* and provides access to a numbered sequence of elements from that array.

A slice literal is declared just like an array literal, except you leave out the element count: `letters := []string{"a", "b", "c", "d"}`

A slice can be created with the built-in function called make, which has the signature: `func make([]T, len, cap) []T`, e.g., `make([]int, 50, 100)`

https://blog.golang.org/go-slices-usage-and-internals

defer

Execution is deferred to the moment the surrounding function returns:

- return statement,
- reached the end of its function body,
- corresponding goroutine is panicking

Used most often with locks/opens:

- Don't forget to perform the function.
- Close to open/lock

```
f, err := os.Open(filename)
if err != nil {
    return "", err
}
defer f.Close()
        … process file …
```

for

```
// Like a C for
for init; condition; post { }

// Like a C while
for condition { }

// Like a C for(;;)
for { }
```

```
for key, value := range oldMap {
    if !key.expired() {
        newMap[key] = value
    }
}
```

if (with a short statement)

```
func pow(x, n, lim float64) float64 {
    if v := math.Pow(x, n); v < lim {
        return v
    }
    return lim
}
```

switch...

```
switch os := runtime.GOOS; os {
case "darwin":
    fmt.Println("OS X.")
case "linux":
    fmt.Println("Linux.")
default:
    // freebsd, openbsd,
    // plan9, windows...
    fmt.Printf("%s.", os)
}
```

… switch …

```
switch t:= time.Now(); {
case t.Hour() < 12:
    fmt.Println("Good morning!")
case t.Hour() < 17:
    fmt.Println("Good afternoon.")
default:
    fmt.Println("Good evening.")
}
```

… switch

```
switch i := x.(type) {
case nil:
    printString("x is nil") // type of x (interface{})
case int:
    printInt(i)             // i is type of int
case float64:
    printFloat64(i)         // i is type of float64
default:
    printString("don't know the type")
}
```

select

A select statement chooses which of a set of possible send or receive operations will proceed.

```
var c1, c2 chan int
var i1, i2 int
select {
case i1 = <-c1:
    print("received ", i1, " from c1\n")
case c2 <- i2:
    print("sent ", i2, " to c2\n")
default:
    print("no communication\n")
}
```

Functions...

Functions:

- Are types.
- Function declarations associate a name to a function body
  - `func IndexRune(s string, r rune) int { … function body ... }`
  - `func flushICache(begin, end uintptr)  // external`
- May have variadic parameters
  - `func Fprintf(w io.Writer, format string, a ...interface{}) (n int, err error)`

Functions do ***not*** manipulate state … the same input *always* produces the same output.

Functions

Methods:

- Functions with receivers are called methods.
- Associates method body with receiver's base type
  - `func (c *Client) Get(url string) (resp *Response, err error)`

Methods are often used when state manipulation is required.

packages

should be same as directory name

std lib > 100 packages

main is special (must have a main)

must import exactly what you need -- more or less will not compile

import must follow package statement

goimports (gofmt)

Packages ...

Packages:

- Go programs are constructed by linking packages together.
- Packages consist of one or more source files.
- All constants, types, variables, and functions are accessible to all files in package.
- main package is special -- all applications must have a main package
- Must import exactly what you need -- more or less will not compile
- import must follow package statement

… Packages …

Package initialization:

- Variables are initialized in declaration order but after any variables they depend on.
- One or more no argument, no return functions named init. Executed in order in a single goroutine.

Program execution:

- Single, unimported package called main with a function that takes no arguments and returns no value.

… Packages

Importing Packages:

- `import "lib/math"   // use math.Sin`
- `import m "lib/math" // use m.Sin`
- `import . "lib/math" // use Sin`
- `import ( … )`
- `import "github.com/yuin/gopher-lua"`

Concurrency - Goroutines

A "go" statement starts the execution of a function call as an independent concurrent thread of control, or goroutine, within the same address space.

```
go <Expression>
```

The expression must be a function or method call; it cannot be parenthesized.

They're called *goroutines* because the existing terms—threads, coroutines, processes, and so on—convey inaccurate connotations. A goroutine has a simple model: it is a function executing concurrently with other goroutines in the same address space. It is lightweight, costing little more than the allocation of stack space. And the stacks start small, so they are cheap, and grow by allocating (and freeing) heap storage as required.

https://golang.org/doc/effective_go.html#goroutines

Concurrency - Channels ...

A channel provides a mechanism for concurrently executing functions to communicate by sending and receiving values of a specified element type. The value of an uninitialized channel is nil.

```
chan T            // used to send and receive values of type T
chan<- float64    // can only be used to send float64s
<-chan int        // can only be used to receive ints
```

A new, initialized channel value can be made using the built-in function make, which takes the channel type and an optional capacity as arguments:

```
make(chan int, 100)
```

… Concurrency - Channels…

```
make (c <- int)
...
close(c)
```

Straight read:

```
n <- c  // returns 0
```

Read with error:

```
x, ok := x <- c  // returns 0, nil
```

Read over range:

```
for x := range c  {   // completes after close
...
```

… Concurrency - Channels…

```
make (c <- int)
...
close(c)
```

Straight read:

```
n <- c  // returns 0
```

Read with error:

```
x, ok := x <- c  // returns 0, nil
```

Read over range:

```
for x := range c  {   // completes after close
...
```

Interfaces

An interface type specifies a method set called its interface. A variable of interface type can store a value of any type with a method set that is any superset of the interface.

```
type Block interface {
    BlockSize() int
    Encrypt(src, dst []byte)
    Decrypt(src, dst []byte)
}
```

All types implement the *empty* interface: `interface{}`

Interfaces - https://golang.org/ref/spec#Interface_types

```
type Locker interface {
    Lock()
    Unlock()
}

type ReadWriter interface {
    Read(b Buffer) bool
    Write(b Buffer) bool
}

type File interface {
    ReadWriter  // same as adding the methods of ReadWriter
    Locker      // same as adding the methods of Locker
    Close()
}
```

The Standard Library and more, much more

- Go 1.10: more than 100 packages in stdlib (including builtin and unsafe)
- Documented at https://golang.org/pkg/
- Many Go packages documented at https://godoc.org
  - Hosts documentation for Go packages on Bitbucket, GitHub, Launchpad and Google Project Hosting.

Tools - go command ...

go <command> [arguments]

- build compile packages and dependencies
- clean remove object files and cached files
- doc show documentation for package or symbol
- env print Go environment information
- bug start a bug report
- fix update packages to use new APIs
- fmt gofmt (reformat) package sources
- generate generate Go files by processing source

… Tools - go command

go <command> [arguments]

- get download and install packages and dependencies
- install compile and install packages and dependencies
- list list packages
- run compile and run Go program
- test test packages
- tool run specified go tool
- version print Go version
- vet report likely mistakes in packages

Tools - standalone

- Cgo enables the creation of Go packages that call C code.
- Cover analyzes the coverage profiles generated by "go test -coverprofile"
- Fix finds Go programs that use old features of the language and libraries and rewrites them to use newer ones.
- Fmt formats Go packages, it is also available as an independent gofmt command with more general options.
- Godoc extracts and generates documentation for Go packages.
- Vet examines Go source code and reports suspicious constructs, such as Printf calls whose arguments do not align with the format string.

Resources...

The Go Programming Language: https://golang.org/

Documentation Page: https://golang.org/doc

- A Tour of Go
- How To Write Go
- Effective Go
- FAQ
- Specification

Go Blog: https://blog.golang.org/

Go Talks: https://talks.golang.org/

… Resources…

GoWiki: https://github.com/golang/go/wiki

Exercism: http://exercism.io/languages/go/about

YouTube: numerous GopherCon sessions -- search for golang

Gophers Slack Team: https://gophers.slack.com

Golang Weekly - https://golangweekly.com/

A weekly newsletter about the Go programming language.

… Resources…

Golang weekly - 197

IN BRIEF

A Comprehensive Guide to Publishing Go Libraries TUTORIAL
Covers dependencies, docs, tests, CI, and licensing.
DARA HAYES

Simply Scale Go-libvirt with Code Generation on DigitalOcean TUTORIAL
Use code generation to extend libvirt's extensive API without leaving the comfortable environment of Go.
DIGITALOCEAN SPONSORED

Instrumenting a Go Service for Prometheus TUTORIAL
Includes writing your own custom metrics, as well.
ALEX DZYOBA

Building a JSON API in Go TUTORIAL
CORY FINGER

An Intro to dep: How to Manage Your Go Project Dependencies TUTORIAL
YING KIT YUEN

Writing a Space Invaders Game with Go TUTORIAL
SAU SHEONG CHANG

A Simple Echo Server in Go TUTORIAL
BEN HYRMAN

Advanced Go Debugging with Delve VIDEO
DEREK PARKER

… Resources …

Books:

- **The Go Programming Language**
  by Alan A. A. Donovan, Brian W. Kernighan -- http://a.co/dN2aDoN
- **Concurrency in Go: Tools and Techniques for Developers**
  by Katherine Cox-Buday -- http://a.co/j40WwrH

… Resources

Community:

- **Gopher Slack Team**
  https://blog.gopheracademy.com/gophers-slack-community/
  Gopher Academy
- **StLGo Meetup**
  https://www.meetup.com/StL-Go/
  Charles Sharp & Ciju John

Thanks

slides: http://speakerdeck/csharp