

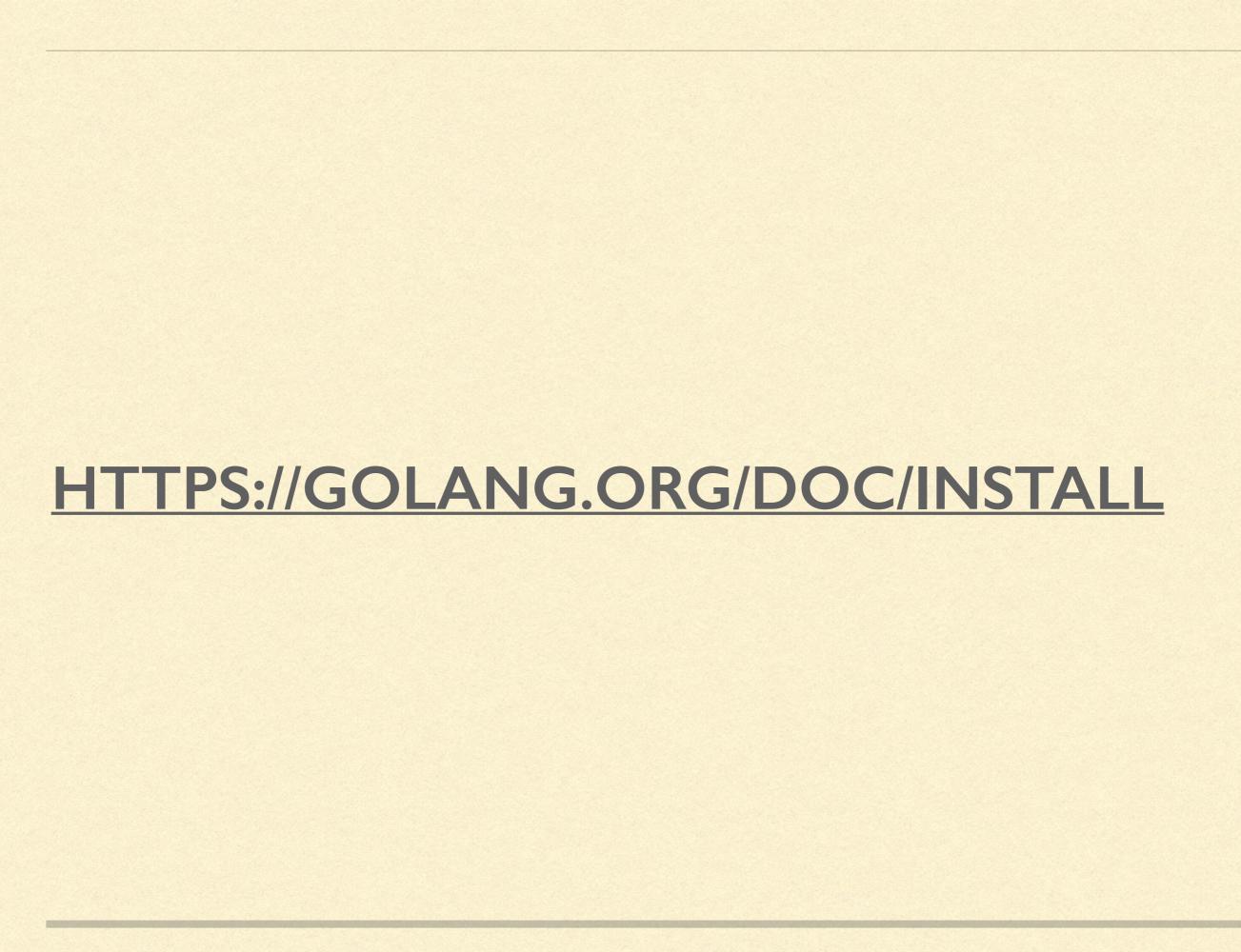
BEGINNING GO WORKSHOP

Sue Spence London Women Who Go

GOTUTORIAL 2

HOWTO INSTALL GO

I. FOLLOW THE INSTRUCTIONS AT GOLANG.ORG



2. CREATE A GO WORKSPACE

WHAT IS A WORKSPACE?

- The Go build tool (toolchain) uses this structure to find, build and maintain your code
- src directory holds all of the project source by package
- obj directory holds all of the package objects
- bin directory contains executable programs

WORKSPACE HOW-TO #1

- Let's call it 'gocode' (or use a name you like)
- mkdir \$HOME/gocode
- cd \$HOME/gocode
- mkdir src pkg bin

WORKSPACE HOW-TO #2

Set up your own package source area

We'll use Github because it's the most common

mkdir -p \$HOME/gocode/src/github.com/<GITHUB_ID>



- Set \$GOPATH Go toolchain needs this
 - export GOPATH=\$HOME/gocode
- Update \$PATH for convenience
 - export PATH=\$PATH:\$GOPATH/bin

4. CHOOSE AN EDITOR

GOOD EDITORS

- Any development-oriented editor is fine
- Vim, Emacs,
- Vim may be the best supported
- Install Atom if you are new to programming
 - https://atom.io/

GO PROGRAM STRUCTURE

There are 25 Go Keywords

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

select

There are 37 Predeclared Names						
CONSTANTS	true	false	iota	nil		
TYPES	int	int8/16/32/64	uint	uint8/16/32/64		
	uintptr	float32/64	complex64/128	bool		
	byte	rune	string	error		
FUNCTIONS	make	len	cap	new		
	append	copy	complex	real		
	imag	panic	recover			

BASIC PROGRAM STRUCTURE

- Each file starts with a package declaration
- Next come imports
- Then "package level" declarations
 - → types, variables, constants or functions

HELLO PEOPLE

```
package main

import "fmt"

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

MORE BASIC RULES

- Package 'main' creates a stand-alone program
- All other package names create a library
- You must import only the packages you need
- Imports must follow the package statement
- Semicolons are not required at the end of a line

DECLARATIONS

A declaration names a program entity and specifies some

or all of its properties

DECLARATIONTYPES

var - variable

const - constant

type - entity type

• func - function

EXAMPLE

```
package main
import "fmt"
const boilingF = 212.0
func main() {
 var f = boilingF
  var c = (f - 32) * 5/9
 fmt.Printf("Boiling point = %g°F or %g°C\n", f, c)
```

VAR

- A var declaration creates a variable of a particular type, gives it a name and sets its initial value
- var temperatureF float = 98.6
- var numMonkeys int = 9

SHORT HAND DECLARATION



- Convenient and 'dynamic'
- temperatureC := 10.0
- freq := rand.float64 *
 2.0
- \bullet i, j := 0, 1

CONSTANTS

- Useful when you want to declare a value that cannot change
- Can be in any scope package or local
- const theSecret = 42

THE 'NEW' FUNCTION



- p := new(int)
- Exactly the same as var p int
 or p := 0

LIFETIME OF VARIABLES

- Package level variables (outside of functions) exist for the entire time the program is running
- All other variables are local and are created when they are declared
- Local variables may be destroyed when no longer reachable

BASIC DATATYPES

- Integers
- Floating-point Numbers
- Complex Numbers
- Booleans
- Strings
- Constants

INTEGERS

- These are the counting numbers
- int, int8, int16, int32, int64
- uint, uint8, uint16, uint32, uint64
- rune is a synonym for int32 ... a unicode code point (地)
- **E.g.** 1, 2, 3, 1005, 9993429394

FLOATING-POINT

- float32, float64
- **1**0.4, 248.638, 112.0,2993884840.35
- Be careful! These variables are tricky to use in calculations.

COMPLEX NUMBERS

- complex64, complex128
- E.g. 2i, 5 + 3i
- Not all that commonly used (unless you're calculating fractals)

BOOLEANS

- bool
- Either true or false

STRINGS

- Immutable sequence of bytes
- "hello, world", "whassup"
- The "strings" package is very useful

CONSTANTS

- Expressions with a value set at compile time, not run time
- Underlying type is a string, number or boolean
- "hello, world", 101.5, true

COMPOSITE DATATYPES

- Arrays
- Slices
- Maps
- Structs

ARRAYS

- Fixed length sequence of 0 or more elements of a type
- Underlying type is string, numeric or composite type
- var alpha [3]int // array of 3 integers

SLICES

- Variable length sequence where elements have the same type
- Strongly connected to arrays and used more often
- Underlying type is string, numeric or composite type
- Looks like an array without the size
- nums := []int{ 0, 1, 2, 3, 4 }

MORE ABOUT SLICES

- You can use 'make' to create them: make([]thing, 10)
- make creates an anonymous array for the slice

MAPS

- Hash table
- Key/Value pairs where all keys are distinct
- Created via the 'make' function
- E.g. ages := make(map[string]int)
- Sample assignment: ages ["Georgia"] = 28
- Elements can be removed with 'delete'
- delete(ages, "Georgia")

STRUCTS

- Aggregate type that groups zero or more named values of arbitrary types in a single entity
- Each value in a struct is called a field
- Sample declarations and assignment for a Film data structure:

```
type Film struct {
   Title string
   Year int
   Director string
   Actors []string
}
var film Film
film.Title = "Teenage Mutant Ninja Turtles"
```

FUNCTIONS

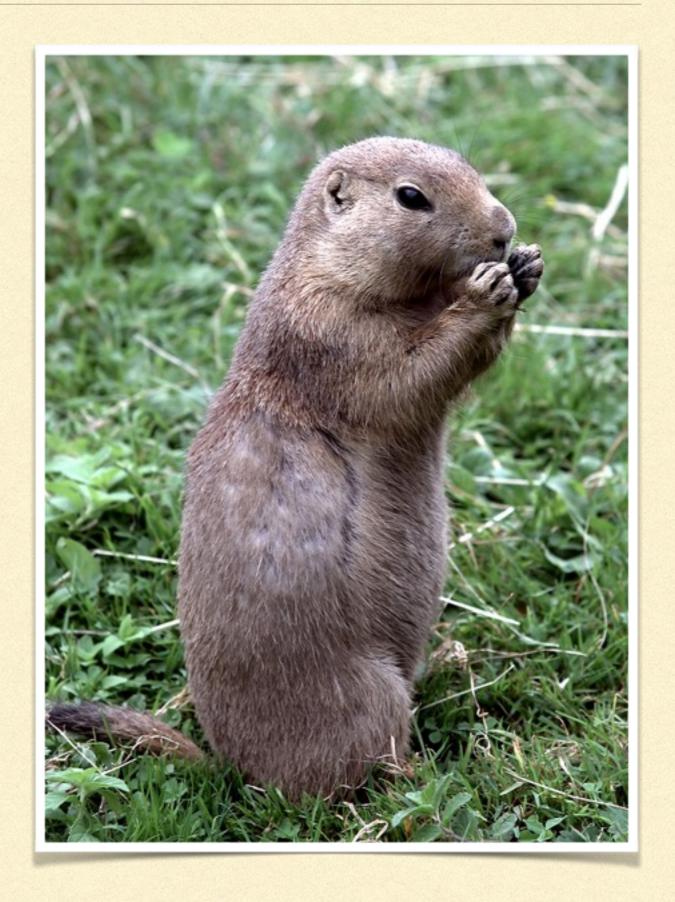
- A sequence of statements that can be called multiple times in a program
- Function declarations have a name, parameter list, optional list of results, and a body

```
func name(parameter-list) (result-list) {
  body
```

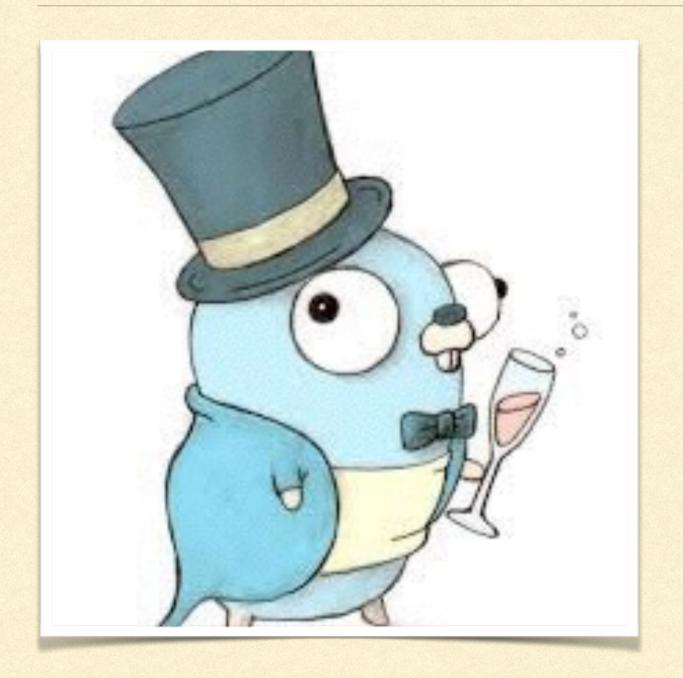
SIMPLE FUNCTION EXAMPLES

- func add(x, y int) int { return x+y }
- func first(x int, int) int { return x }

Time to Try Some Code EXERCISM.IO



WHEW



That's all for now.

