

*Women Who Go
meeting 1.0*

introductions

Who you are? Career?
Programming experience?
Why you use or want to learn Go?

Let's make a boring presentation
much more boring
with writing
'Introduction'
as the title of
introduction
slide



som^{ee}cards
user card

agenda

- Why use / learn Go?
- Lab: Install Go & 'Hello World'
- Have fun!



why learn Go?



*many projects/companies
using Go!*



- simple by design
- great tooling
- features to help you build fast
 - GC (Garbage Collection)
 - concurrency (goroutines/channels)

easy for teams to learn & be productive in
quickly

simple by design.

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

few keywords.

```
err := s.Update(v)
if err != nil {
    // handle error
}
```

explicit errors.

```
package main

import "fmt"

// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```


`package main` how to define a package in Go.

```
import "fmt"
```

```
// swap will swap the variables
```

```
func swap(x, y string) (string, string) {  
    return y, x  
}
```

```
func main() {  
    var hello string = "hello"  
    a, b := swap(hello, "world")  
    fmt.Println(a, b)  
}
```

```
package main

import "fmt"

// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```

```
package main
```

```
import "fmt" how to import an external package.
```

```
// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}
```

```
func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```

```
package main

import "fmt"

// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```

```
package main
```

```
import "fmt"
```

```
// swap will swap the variables
```

```
func swap(x, y string) (string, string) {  
    return y, x  
}
```

this is a function definition.

```
func main() {  
    var hello string = "hello"  
    a, b := swap(hello, "world")  
    fmt.Println(a, b)  
}
```

```
package main

import "fmt"

// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```

```
package main
```

```
import "fmt"
```

```
// swap will swap the variables
```

```
func swap(x, y string) (string, string) {  
    return y, x  
}
```

```
func main() { the program entry point
```

```
    var hello string = "hello"
```

```
    a, b := swap(hello, "world")
```

```
    fmt.Println(a, b)
```

```
}
```

```
package main

import "fmt"

// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```



```
package main
```

```
import "fmt"
```

```
// swap will swap the variables
```

```
func swap(x, y string) (string, string) {  
    return y, x  
}
```

```
func main() {
```

```
    var hello string = "hello"  
    a, b := swap(hello, "world")  
    fmt.Println(a, b)
```

```
}
```

declaring variables
(2 ways)

```
package main

import "fmt"

// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```

```
package main

import "fmt"

// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}
```

```
func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```

calling a function in another package.

```
package main

import "fmt"

// swap will swap the variables
func swap(x, y string) (string, string) {
    return y, x
}

func main() {
    var hello string = "hello"
    a, b := swap(hello, "world")
    fmt.Println(a, b)
}
```

tooling!

```
$ go run main.go  
world hello
```

```
$ go build -o hello.go  
$ ./hello  
world hello
```

running/building code.

```
$ gofmt main.go  
package main
```

```
import "fmt"  
  
func swap(x, y string)  
(string, string) {
```

enforcing conventions.

```
$ go get -v github.com/  
mailgun/godebug  
github.com/mailgun/godebug  
(download)
```

getting code.

```
$ go test -v .  
=== RUN TestThis  
--- FAIL: TestThis (0.00s)  
    apple_test.go:6:  
FAIL  
exit status 1
```

testing code!

- statically typed
- statically compiled binaries
- opinionated code conventions/formatting and supporting tooling

grows with the team and project

great production story

types in Go

```
var hello string // declare a string  
hello = "hello"  // assign it to a string
```

or

```
hello := "hello" // it's inferred to be a string
```

static types.. but with type inference.

types in Go

basics

bool string int int8 int16 int32 int64 uint uint8
uint16 uint32 uint64 uintptr byte rune float32 float64
complex64 complex128

arrays, maps,
slices

```
// arrays have a fixed size  
var nums []int = {1, 2, 3}
```

```
// slices are dynamic  
nums := make([]int, 5)  
nums = append(nums, 1)
```

```
// maps are like hashtables or dicts  
stringmap := make(map[string]string)
```

no generics, no “magic”

strong conventions

tabs not spaces.

curly braces with the func definition

CamelCase enforced.

exported functions/variables
are capitalized.

all package files in same folder.
opinionated workspace.

**= your team writing code in the same
way.**

opinionated workspace

- Go workspaces are defined by your `$GOPATH` env variable.
- `$GOPATH/bin` - binaries installed here
- `$GOPATH/pkg` - packages built here
- `$GOPATH/src` - sources here

```
.
├── bin
│   └── helloworld
├── pkg
└── src
    ├── github.com
    │   └── jandre
    │       └── helloworld
    │           └── main.go
```

- comprehensive, well-designed stdlib
- tooling for importing external packages
- many robust open source projects built with Go

strong, growing community

lots of code to build from

resources!

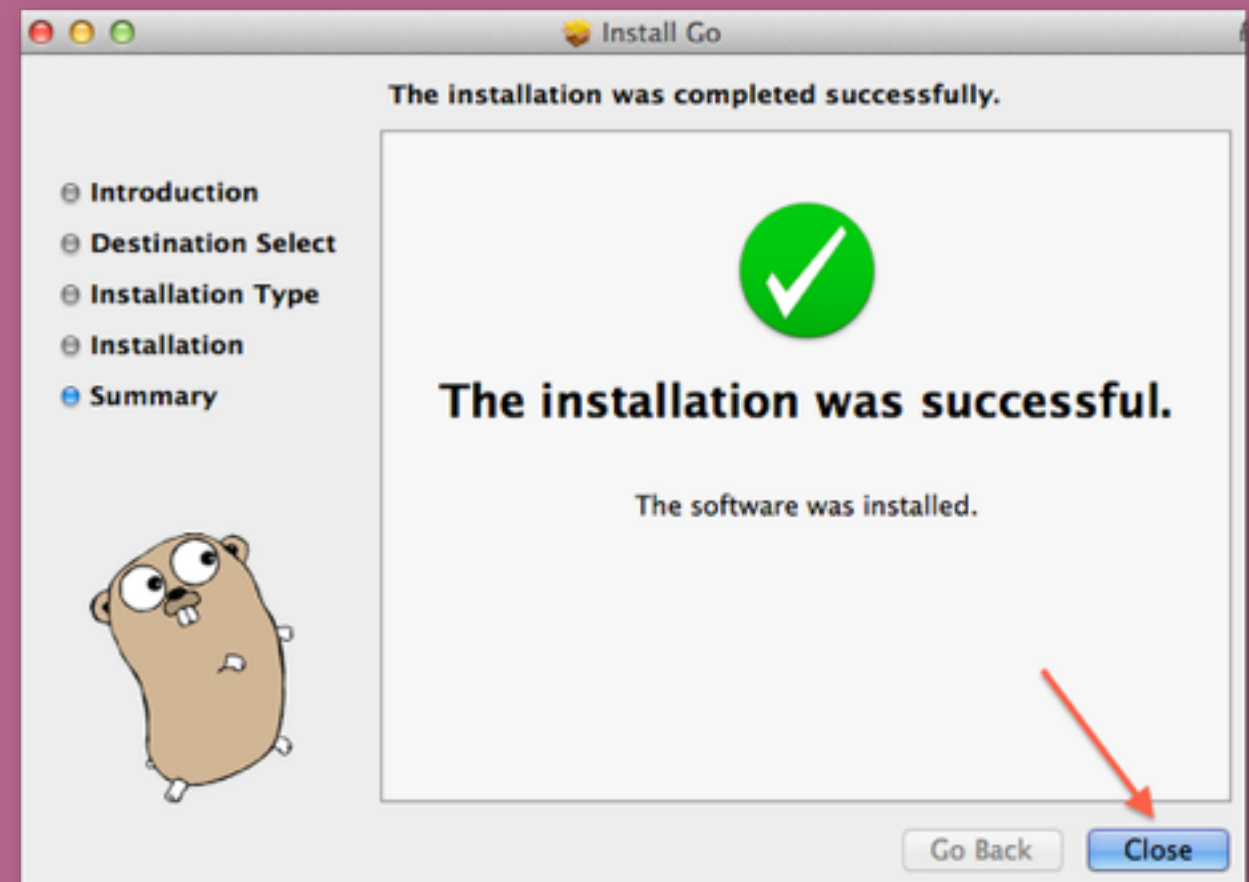
- A tour of Go: <https://tour.golang.org/welcome/1>
- Go by example: <https://gobyexample.com>
- Boston GoBridge workshop 2/6: <https://www.bridgetroll.org/events/237>
- Boston GoLang meetup (next week!): <http://www.meetup.com/Boston-Go-lang-User-Group/events/227889016/>

lab!



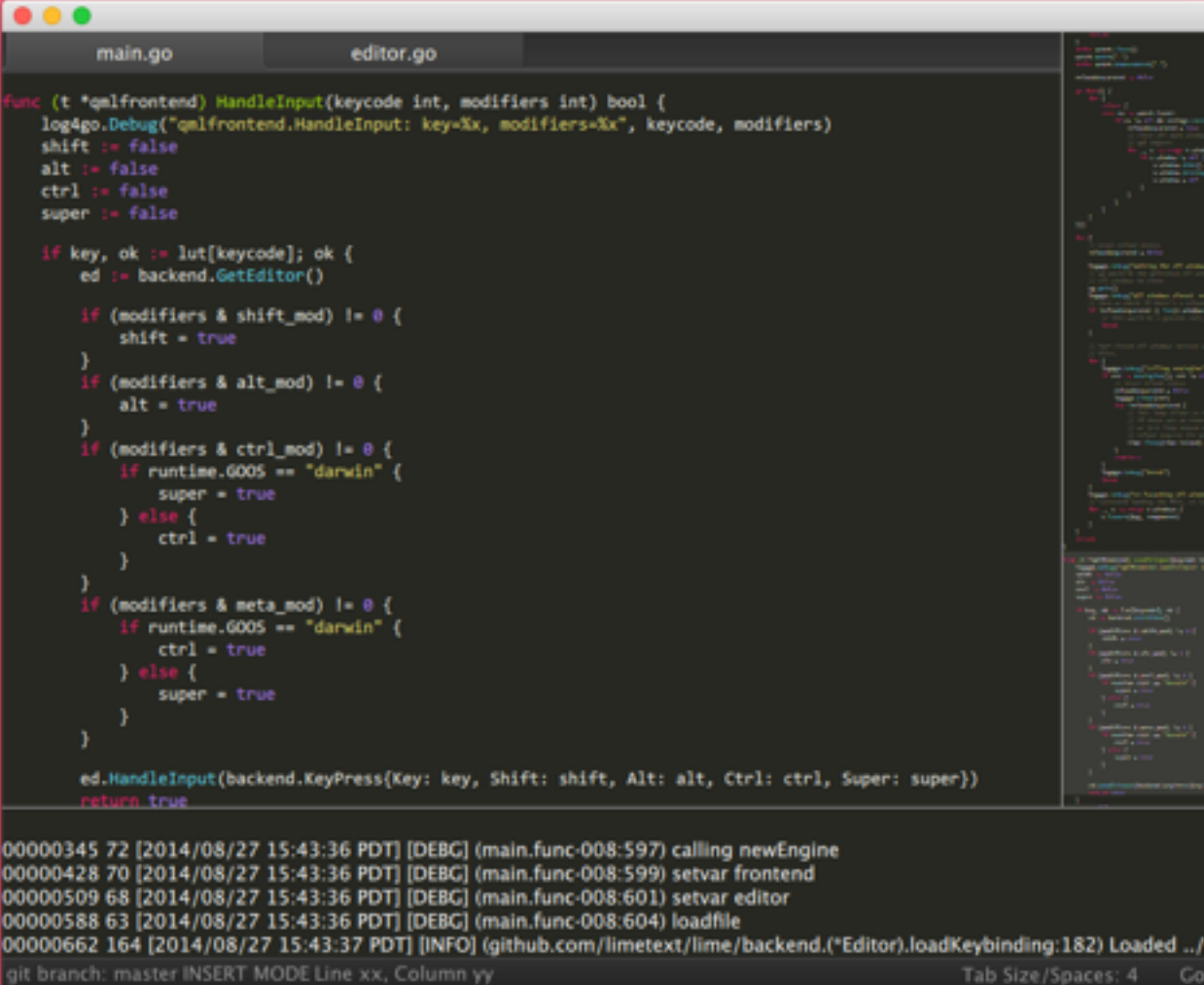
1. install Go

- Windows: <https://golang.org/dl/>
- Mac: <https://golang.org/dl/> or Homebrew (<http://brew.sh/> using `brew install golang`)
- Linux: <http://ask.xmodulo.com/install-go-language-linux.html>



2. Install code editor support

- Sublime: <https://github.com/DisposaBoy/GoSublime>
- Atom: <https://atom.io/packages/go-plus>
- VIM: <https://github.com/fatih/vim-go>



The screenshot shows a Go IDE window with two tabs: 'main.go' and 'editor.go'. The 'main.go' tab is active, displaying Go code for a function 'HandleInput'. The code defines variables for shift, alt, ctrl, and super keys, and uses a lookup table 'lut' to determine if a key is pressed. It then calls 'backend.GetEditor()' and 'ed.HandleInput()' to handle the key press. The 'editor.go' tab is also visible, showing more code. At the bottom of the window, a debug console displays log messages with timestamps and function names, such as 'calling newEngine', 'setvar frontend', 'setvar editor', 'loadfile', and 'loadKeybinding'. The status bar at the bottom indicates 'git branch: master INSERT MODE Line xx, Column yy', 'Tab Size/Spaces: 4', and 'Go'.

```
func (t *qmlfrontend) HandleInput(keycode int, modifiers int) bool {
    log4go.Debug("qmlfrontend.HandleInput: key=%Xx, modifiers=%Xx", keycode, modifiers)
    shift := false
    alt := false
    ctrl := false
    super := false

    if key, ok := lut[keycode]; ok {
        ed := backend.GetEditor()

        if (modifiers & shift_mod) != 0 {
            shift = true
        }
        if (modifiers & alt_mod) != 0 {
            alt = true
        }
        if (modifiers & ctrl_mod) != 0 {
            if runtime.GOOS == "darwin" {
                super = true
            } else {
                ctrl = true
            }
        }
        if (modifiers & meta_mod) != 0 {
            if runtime.GOOS == "darwin" {
                ctrl = true
            } else {
                super = true
            }
        }
    }

    ed.HandleInput(backend.KeyPress{Key: key, Shift: shift, Alt: alt, Ctrl: ctrl, Super: super})
    return true
}
```

00000345 72 [2014/08/27 15:43:36 PDT] [DEBUG] (main.func-008:597) calling newEngine
00000428 70 [2014/08/27 15:43:36 PDT] [DEBUG] (main.func-008:599) setvar frontend
00000509 68 [2014/08/27 15:43:36 PDT] [DEBUG] (main.func-008:601) setvar editor
00000588 63 [2014/08/27 15:43:36 PDT] [DEBUG] (main.func-008:604) loadfile
00000662 164 [2014/08/27 15:43:37 PDT] [INFO] (github.com/limetext/lime/backend.(*Editor).loadKeybinding:182) Loaded ...
git branch: master INSERT MODE Line xx, Column yy Tab Size/Spaces: 4 Go