Brief introduction to Go

Who is this guy?

- Mark Wolfe <mark@wolfe.id.au>
- @wolfeidau on twitter
- github.com/wolfeidau
- CTO at Ninja Blocks http://www.ninjablocks.com/

So what is go?

- Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.
- Statically compiled, garbage collected language
- Statically typed, with Duck Typing (interfaces)

So what is go?

- Supports multi-core
- Concurrency primitives
- Closures
- Somewhere between C and Python

The go way

- Very opinionated
- gofmt, mixed case, Capitalisation controls privacy
- Simple is better than complicated
- Compiler does the heavy lifting
- Search for golang

Hello World

```
package main

import (
   "fmt"
)

func main() {
   fmt.Println("yo")
}
```

Building Hello World

```
Of course you read the README.md..

Navigate to the contents of gostick in your home.

tar xvzf packages/go1.2.darwin-amd64-osx10.8.tar.gz

source env.sh

cd helloworld

go build -v

./helloworld
yo
```

Go in practice

- Go is particularly suited to building network services.
- Standard library is quite small.
  - Largest packages are crypto, encoding and net
- Ecosystem is still quite young, lots of gaps

Echo Server

- Bind to a TCP port and wait for connections
- Accept connections
- Read Text from the connection
- Write Text back to the connection

Echo Server

- **Bind to a TCP port and wait for connections**
- Accept connections
- Read Text from the connection
- Write Text back to the connection

Bind to a TCP port and wait for connections

```
package main

import (
   "log"
   "net"
)

func main() {

   log.Printf("Listening")
   listener, err := net.Listen("tcp", ":8000")
   if err != nil {
      log.Fatalln(err)
   }
}
```

Variable Declaration

- long declaration

```
var foobar unint64
```

- short declaration, type is inferred automatically

```
listener, err := net.Listen("tcp", “:8000")
```

- This statement declares two variables, listener and err.

- Go strives to save on typing when the compiler can figure it out

Error Handling

- Uses multiple return values to return errors
- You will write this a lot.

```
if err != nil {
   // do something
}
```

- Yes this gets verbose, break your code up

Echo Server

- Bind to a TCP port and wait for connections
- **Accept connections**
- Read Text from the connection
- Write Text back to the connection

Accept connections

```
func main() {

   log.Println("Listening")
   listener, err := net.Listen("tcp", ":8000")
   if err != nil {
      log.Fatalln(err)
   }

   for {
      client, err := listener.Accept()
      if err != nil {
         continue
      }
      handleClient(client)
   }

}
```

Echo Server

- Bind to a TCP port and wait for connections
- Accept connections
- **Read Text from the connection**
- **Write Text back to the connection**

Client Handler

- The connection handler

```
func handleClient(client net.Conn) {
   for {
      // read from client
      // write to client
   }
}
```

- Reading from a connection

```
func Read(b []byte) (int, error)
```

- So what is `byte[]`?

Primitive Go Types

- All of the normal candidates, `byte`, `uint`, `float`
- Built in String type (Unicode and immutable)
- `int` and `uint` are architecture width
- `byte` is a synonym for `uint8`
- `rune` is a synonym for `uint32`

More Types

- arrays: of a declared, fixed length
- slice: a segment “slice” of an array
- map: key/value storage
- pointer: much like C (uses & and *)
- more later

Arrays

- Declare an Array

```
var a [4]int
a[0] = 1
i := a[0]
// i == 1
```

- Slices

```
letters := []string{"a", "b", "c", "d"}
func make([]T, len, cap) []T
var s []byte
s = make([]byte, 5, 5)
// s == []byte{0, 0, 0, 0, 0}
```

Echo Server

- Bind to a TCP port and wait for connections
- Accept connections
- **Read Text from the connection**
- **Write Text back to the connection**

Client Handler

```
func handleClient(client net.Conn) {
  for {
    buf := make([]byte, 4096)
    // read from client
    numbytes, err := client.Read(buf)
    if numbytes == 0 || err != nil {
      return
    }
    // write to client
    client.Write(buf)
  }
}
```

Building Echo Server

```
cd echo

go build -v

./echo
2014/02/02 08:40:17 Listening

nc localhost 8000
```

Packages

- Lets write a library called strings
- We'll use github.com/user as our base path.
- Reverse a string
- With a test

Strings library

```go
package strings

// Reverse the string passed in taking into
// consideration double width characters
func Reverse(str string) string {
   n := len(str)
   runes := make([]rune, n)
   for _, rune := range str {
      n--
      runes[n] = rune
   }
   return string(runes[n:])
}
```

Strings Test

```
package strings

import (
   "testing"
)

func TestReverse(t *testing.T) {
   const in, out = "Hello, 鸡炒饭", "饭炒鸡 ,olleH"
   if x := Reverse(in); x != out {
      t.Errorf("Reverse(%s) = %s, expected %s", in, x, out)
   }
}
```

Building Strings

```
cd gopath/src/github.com/user/strings

go test
PASS
ok    github.com/user/strings 0.007s
```

The GOPATH

- Commonly referred to as the workspace
- Staging area for code retrieved using `go get`
- Contains `src` `pkg` `bin` directories
- This is the recommended place to work on your code
- For us `$PWD/gopath` is exported as `GOPATH`

Use the github.com/user/strings Library

```
package main

import (
    "github.com/user/strings"
    "log"
    "net"
)

...

func handleClient(client net.Conn) {
    for {
        buf := make([]byte, 4096)
        // read from client
        numbytes, err := client.Read(buf)
        if numbytes == 0 || err != nil {
            return
        }
        rev := strings.Reverse(string(buf))
        // write to client
        client.Write([]byte(rev))
    }
}
```

Building Echo Reverse Server

```
cd echoreverse

go build -v

./echoreverse
2014/02/02 08:40:17 Listening

nc localhost 8000
```

Summary

- Go is a pretty simple language to use
- Very opinionated, do it their way..
- Easy to learn
- Great for operations
- Solid foundation for network services

What Next?

- Read some code.

- http://www.somethingsimilar.com/2013/12/27/code-to-read-when-learning-go/

- Hack on open source projects
  - Docker https://github.com/dotcloud/docker
  - Packer https://github.com/mitchellh/packer
  - Serf https://github.com/hashicorp/serf

The End

- Questions?