Go for the Rubyist

Dave Grijalva
Director of Platform Technology
ngmoco:)

twitter/github: @dgrijalva
dave@ngmoco.com

What is Go?

- A new programming language from Google
- C-like syntax
- Statically compiled
- Statically typed
- Interface oriented
- Concurrent
- Garbage collected
- Memory safe
- Fast

Multiple Assignment

```
// exchange a and b

a, b = b, a

// multiple return values

file, err := os.Open(“myFile.txt”)

// ignore unneeded values

string, _ = json.Marshal(myData)
```

Types

```
type myInt int

type myStruct struct {
  stuff, things string
}

type myFuncType func(a string)bool

type myStructPtr *myStruct
```

Methods

```
// Any type can have methods
type MyType int
func (i MyType) String()string {
  return fmt.Sprintf(“%v”, i)
}

type FooType struct{a, b string}
func (f *FooType) String()string {
  return fmt.Sprintf(“%v foo %v”, f.a, f.b)
}
```

Interfaces

```
// Declare an interface
type MyThingie interface {
  Foo(string)int
}

// Implement an interface
type MyType struct{a, b string}
func (m *MyType) Foo(string)int {
  // MyType satisfies interface MyThingie
  return 0
}
```

Interfaces

```
// All types satisfy the empty interface

var anything interface{}

// Lots of things are io.Readers, including os.File and
net.Conn

var myReader io.Reader
myReader, err = os.Open(“myFile.txt”)
myReader, err = net.Dial(“google.com:80”)
```

Type Assertions

```
// Basic type checking
if myThing, ok := someVar.(MyThingie); ok {
  myThing.Foo(“stuff”)
}

// Type switches
switch myThing := someVar.(type) {
case string: fmt.Println(myThing)
case MyThinie: myThing.Foo(“stuff”)
default: fmt.Println(“I dunno”)
}
```

Let’s get concurrent

goroutines

```
// Do something
for i := 0; i < n; i++ {
  DoSomething(i)
}


// Do something concurrently
for i := 0; i < n; i++ {
  go DoSomething(i)
}
```

goroutines

```
// Must be a function call, but funcs can be inline
go func(){
  // go do something
}()

// Works with methods too
go myVar.DoStuff(abc)
```

“Don’t communicate by sharing memory.
Share memory by communicating.”

Channels

```
// A typed queue
c := make(chan int)

// Buffered or unbuffered
unbuf := make(chan int)
buf := make(chan int, 100)

// Send to a chan
c <- 1

// Receive from a chan
i := <-c
```

Select

```
// Receive from multiple chans
// Exactly one will succeed
select {
case i := <-myChanA:
  fmt.Println(“Received from chan A”, i)
case i := <-myChanB:
  fmt.Println(“Received from chan B”, i)
}
```

Non-Blocking

```
// Non-blocking receive
select {
case i := <-myChan: fmt.Println("Received from chan", i)
default: fmt.Println("Chan is empty")
}

// Non-blocking send
select {
case myChan <- i: fmt.Println("Sent to chan", i)
default: fmt.Println("Chan is full")
}
```

Timeouts

```
// Timers send triggers using chans
select {
case i := <-myChanA:
  fmt.Println("Received from chan A", i)
case i := <-time.After(1e9):
  fmt.Println("Timeout")
}
```

Demo

Ruby

- Ruby 1.9 Fibers
- Agent (github.com/igrigorik/agent)
- Revactor (revactor.github.com)

Demo

ngmoco:) is hiring!

ngmoco.com/careers

Questions?

Example code at: github.com/dgrijalva/gogaruco2011