Intro to Go

Where did it come from, where did it Go?

Admin

- Syllabus
  - Anyone new this class? Get a Syllabus
- Any trouble installing the tool chains?

Go headline features

- Go (Sometimes calls Golang)
  - Headline features:
    - Compiled
    - Statically typed
      - Variable will always refer to same type of value
    - Structurally Typed
      - Type equivalence by definition not name
    - Memory safe
      - No buffer overflows, unsafe pointer operations
    - Garbage collected
    - Focus on concurrency
      - One of Go's claims to fame

Go Pedigree

- Came from google
  - Just as java came from Sun(Oracle)
  - Originally by
    - Robert Griesemer, Rob Pike, and Ken Thompson
- Open source
  - https://go.googlesource.com/go
    - git clone https://go.googlesource.com/go
  - “BSD-style” licence
    - https://golang.org/LICENSE

Go A First impression

- From original go book (Donovan and Kernighan)– chapter 1 (not updated since 2016)
  - package main
  - import (
    - "fmt"
    - "io/ioutil"
    - "net/http"
    - "os"
    -)
  - func main() {
    - for _, url := range os.Args[1:] {
    - resp, err := http.Get(url)
    - if err != nil {
      - fmt.Fprintf(os.Stderr, "fetch: %v\n", err)
      - os.Exit(1)
    - }
    - b, err := ioutil.ReadAll(resp.Body)
    - resp.Body.Close()
    - if err != nil {
      - fmt.Fprintf(os.Stderr, "fetch: reading %s: %v\n", url, err)
      - os.Exit(1)
    - }
    - fmt.Printf("%s", b)
    - }
  - }

Go A First Impression

- When I first looked at Go
  - It looked like python and C++ had a baby
  - Of course I don't know algol
- Python philosophy :
  - There is one 'right' way to do things
  - (harder to see recently)
  - This is pythonic, but not enforced by compiler/interpreter.
- With go – often is enforced by compiler

Good Style

- Go ‘Good Style’ is often compiler enforced
  - Unused variables are a compiler error
  - Unused imports are a compiler error
- Online flame wars are often about what “good style” is
- Go often settles these by making the compiler only work for the approved style
- Your book explains some of this in a little more depth in chapter 1 (eg page 7)

gofmt

- Go helps you be compile ready with gofmt
  - Can run on command line
    - Or let goland do it for you.
  - Gofmt pronunciation
    - Do you go with the majority?
    - Or with the crusading minority?
  - Gofmt sort of like python-black
    - Simply rewrites your code to be ‘proper’ (idiomatic) go

So Go, Syntactically

- So lets take a “brief” look at how go implements most of the programming constructs you’ll need

Comments

- Comments are really useful when learning a language
- Comments in Go same as C++
  - Go took them just like java did
  - // line comments
  - /*
  - Multiline comments
  - */
  -

Variables

- Variables are statically typed, but type can be inferred
  - var name string //creates a new variable called name of type string with an empty string
  - var name2 = “Imelda” //creates a new variable called name2 of type string with initial value “Imelda”
  - var num1, num2 int = 100, 300
  - var3 :=3.14159

Types

- Go, like java, has distinction between basic types and all other types
- Basic types:
  - Boolean
  - string
  - and number (several number types)
    - uint8, uint16, uint32, uint64, int8, int16, int32 and int64, etc see chap 2 in learning go book.

Constants

- Two slides ago – var3 was clearly what?

Constants

- Two slides ago – var3 was clearly what?
- Pi right? So it really shouldn't be a variable
- Constants in go, more like C++ than python
  - const pi = 3.14159 //math.Pi is better
  - can't be changed
  - Notice constant is lower case
    - Most languages have upper case.
    - Why? (class discussion)

Go Project is a package

- Your go project is a package
  - You remember from last time, main package is the one that runs first.
- If you want to use a function or variable in C/C++ from one file to another how do you do it?

Go Project is a package

- Your go project is a package
  - You remember from last time, main package is the one that runs first.
- If you want to use a function or variable in C/C++ from one file to another how do you do it?
  - Put the declaration in a header and import the header right? (then link everything of course)
- How about python?

Go Project is a package

- Your go project is a package
  - You remember from last slide set, main package is the one that runs first.
- If you want to use a function or variable in C/C++ from one file to another how do you do it?
  - Put the declaration in a header and import the header right? (then link everything right) maybe make public
- How about python?
  - pip install and then import. No public/private needed
- Java?

Go exported symbols

- If you want to use something* outside of its module in go
  - *Function, variable, class etc
- Name that with a first letter capitalized.
  - Names beginning with caps are exported
  - Names beginning with lowercase are not.
    - There now I've saved you an hour of banging your head against your keyboard

Functions

- Create a function in go using keyword func,
  - func <function name>(<param list>) <ret type>{
    - <function body>
    - }
  - A few things to look at here:
    - Return type is after param list (unlike java/c/C++, but like python/swift)
    - Param list can be empty, when not, param name first then type
    - And that opening brace? It must be there. Compile error for being on next line.
      - Avoids one of the favorite java flame wars

Example Function

- A simple example function from the golang tour
  - func add(x int, y int) int {
    - return x + y
  - }
  -

Multi value returns

- Some language (eg python, lisp) support multi value returns
  - Mostly interpreted languages
- Go embraces multi-value returns and really uses it.
- From the http standard library:
- func Get(url string) (resp *Response, err error) {
  - return DefaultClient.Get(url)
- }
- resp is a Response pointer, second return value as error is the go way.

Functions II

- A function using http.Get

```
package main

import (
   "fmt"
   "log"
   "net/http"
)
func main() {
   response, err := http.Get("https://news.ycombinator.com/")
   if err != nil{
   log.Fatal(err)
   }
   defer response.Body.Close()
   fmt.Print(response.Body)
}
```

- A few things:
  - First the multiple returns are captured by two new variables
  - The err is checked first, then ignored.
    - No exceptions - discuss

Imports

- In previous slide imports to use code from other packages
  - And their exported symbols
- Import a single package
  - import “fmt”
- More commonly, import multiple packages

```
import (
    "fmt"
    "log"
    "net/http"
)
```

- Important an unused library is a compile error
  - gofmt to the rescue – run automatically by goland

Selection

- Selection in Go (AKA if)
  - if <condition/Boolean>{
    - <do this if true>
    - }
- Or
  - if <condition/Boolean>{
    - <do this if true>
    - }else{
    - <do this if false>
    - }
  - No parens around condition, but must have braces {} around body even if one line

Statements

- How does a java statement end?

Statements

- How does a java statement end?
  - ;
  - Same as c/c++/C# and other “c-like” languages\
- How does a python statement end?

Statements

- How does a java statement end?
  - ;
  - Same as c/c++/C# and other “c-like” languages\
- How does a python statement end?
  - With the end of the line except for special circumstances

Go Statements

- How Does a go Statement end?
- Reminder:

```
package main

import (
     "fmt"
     "io/ioutil"
     "log"
     "net/http"
)
func main() {
     response, err := http.Get("https://news.ycombinator.com/")
     if err != nil{
     log.Fatal(err)
     }
     defer response.Body.Close()
     dataAsBytes, err := ioutil.ReadAll(response.Body)
     if err != nil{
     log.Fatal(err)
     }
     fmt.Print(string(dataAsBytes))
}
```

- Code is a mangling of

  https://www.devdungeon.com/content/web-scraping-go

Go Statements

- How Does a go Statement end?
- Reminder:

```
package main

import (
     "fmt"
     "io/ioutil"
     "log"
     "net/http"
)
func main() {
     response, err := http.Get("https://news.ycombinator.com/")
     if err != nil{
          log.Fatal(err)
     }
     defer response.Body.Close()
     dataAsBytes, err := ioutil.ReadAll(response.Body)
     if err != nil{
          log.Fatal(err)
     }
     fmt.Print(string(dataAsBytes))
}
```

- More like python (remember compiler puts ; in for you)
  - End of line except for special circumstances

Selection II

- A more complicated selection example
- if num := 9; num < 0 {
- fmt.Println(num, "is negative")
- } else if num < 10 {
- fmt.Println(num, "has 1 digit")
- } else {
- fmt.Println(num, "has multiple digits")
- }
- Notice two statements in first condition
  - Also variables created in condition are available in all later branches

Repetition

- In programming theory, two types of repetition, definite and indefinite
  - For and while in most languages
- Go has only for – which it uses for both

Basic for

- Basic for loop looks a lot like C-like language for loop

```
func countDown(start int){ //in honor of falcon heavy launch
    for counter := start; counter >0; counter--{
        fmt.Println(counter)
    }
    fmt.Println("blastoff")
}
```

- Again
  - no parens around setup, but required braces
  - Scope of variables created in initialization statement only that for-loop

C-like diversion

- Have you ever seen this in c-like languages?
  - `for(;;)`
  - `{`
  - `//do stuff here`
  - `if (something)`
    - `break;`
  - `}`
- Legal, totally unnecessary these days
  - Why was it common 30-40 years ago?

C-like diversion

- Have you ever seen this in c-like languages?
  - `for(;;)`
  - `{`
  - `//do stuff here`
  - `if (something)`
    - `break;`
  - `}`
- Legal, totally unnecessary these days
  - Why was it common 40ish years ago?
  - Optimizing compilers did better with for than while

For II

- You can omit the initialization and post part of the for (not the condition)
  - makes it functionally what other languages use while
  - `func main() {`
    - `sum := 1`
    - `for ; sum < 1000; {`
    - `sum += sum`
    - `}`
    - `fmt.Println(sum)`
  - `} //From https://tour.golang.org/flowcontrol/2`
  - Semi colons are optional – can be dropped

Forever for's

- Oh look, Go's puns have infected your instructor
- While (true) is spelled differently in go. If we want to loop forever (till break)
- for{
- //Do something forever
- }

Break

- As with most languages Go has break
  - Breaks out of the innermost for, switch or select
    - For we saw.
    - Switch works like C-like switch (mostly)
    - Select we'll defer (all puns intended) on (used like switch but for messages)

Strings

- Strings are officially “an immutable sequence of bytes”
  - Can contain 0 (null byte)
  - Usually interpreted as UTF-8 (unicode)
  - Utf-8 characters are called ‘runes’
  - len(string) returns number of bytes not runes
  - Use utf8.RuneCountInString(<string>) to find out how many characters are in string.
    - See book description (page 50 in the inset box) for the varying number of bytes in a utf-8 Character/rune

Strings II

- Strings can be slices in go just like in python
  - Substring/slices are very fast
  - Because strings are immutable
    - Slices/substrings share same memory as parent string.
- Suppose you want to programmatically build a string from parts
  - From go 1.10 on use strings.Builder
  - Before that either inefficient concate += or write bytes (uggg shades of java 1.2)

String Builder

- Since go 1.10 use strings.Builder to concat strings
- Example:
  - `func join(strs ...string) string {`
  - `var sb strings.Builder`
  - `for _, str := range strs {`
  - `sb.WriteString(str)`
  - `}`
  - `return sb.String()`
  - `}`
- Range iterates over elements in data structures
  - First return value is usually element number

Basics

- Now you know how to do all of CS1 in go
- More Go next time. Assignment next.

Assignment

- Read chapters preface & 1-3 of the Learning Go book.
  - Those of you who were here for the first class already read preface and chapter 1.
  - A mix of easy reading and deeper reading. Plan to take some time for it
  - And play with it. Assignment next time to stretch your gopher wings.