CS 677: Big Data

A Whirlwind Tour of Go

Lecture 3

Today's Schedule

- Thoughts on AI
- Go Basics
- A break (to write some Go)

Today's Schedule

- **Thoughts on AI**
- Go Basics
- A break (to write some Go)

Thoughts on AI

- Before we begin, let's talk about AI… or, more specifically, LLMs.
- When learning a new language, LLMs are a huge help!
  - And when writing a lot of code in a new language? Even better!
- There are no restrictions on writing code with LLMs in this class
  - Via chat, AI agents, etc.
  - But…

Human in the Loop

- Since we're trying to learn, make sure you are fully in control and knowledgeable of your code
- Recommendation 1: practice writing Go by hand if you haven't before. It's slower, but you will remember more.
- Recommendation 2: if you are using an agent, don't run it in YOLO mode. Actually read and understand the changes being made. Revise!
- Recommendation 3: remember, the objective is learning

Good Advice

- Obviously listening to me is a bad idea, so…
- I really like Oxide Computer RFD 576 for guidelines on LLM usage
  - (Link on schedule page)
- It's practical, reasonable, and probably what we need in a world of hot takes

Today's Schedule

- Thoughts on AI
- **Go Basics**
- A break (to write some Go)

Time to Go

- Yes, I am going to make lame jokes like this all semester.
- I mentioned last class we'll be using Go (a.k.a "golang") for our systems-building projects
- Let's talk a bit about Go…

Go

- Go sits somewhere between C and Java in terms of functionality
- **Like C**: compiled, simple syntax, fairly small standard library (at least from a modern perspective), easy to write systems software
- **Like Java**: memory safety, garbage collection, rapid development
- **New**: a different approach to concurrency
  - …that works well for distributed systems

Communicating Sequential Processes (CSP)

- The CSP paradigm is why Go is so great for distributed systems
- Lots of sequential *goroutines* run concurrently
  - Being sequential makes them easy to reason about
- The goroutines communicate via *message passing* instead of using shared memory
  - Ideally that means much less synchronization (mutexes et al.) is needed
- To launch a goroutine, all you need is the `go` keyword!

Pros and Cons

- Pros:
  - Easy to pick up
  - Fairly fast
  - Compiled, so no need to distribute JVMs etc
  - Syntax isn't huge and bulky
- Cons:
  - It's not as mature as Java
  - The standard library lacks some data structures you might expect to have built in
  - Still (generally) not as fast as C, C++, Rust

Learning Curve

- Go is easy to pick up if you're already used to Java, C, or Python
- I'd estimate it takes about 1 week to be productive, 2 weeks to really start getting the hang of it.
- **Lots** of companies are using Go these days
  - Building microservices
  - Building systems software
  - Even modern websites

Obligatory “Hello World”

```
package main

import "fmt"

func main() {
   fmt.Println("Hello world!")
}
```

- `Println` like Java but so easy to type!
- Hmm, no return value or arguments?
- NO SEMICOLONS?!

Compiling and Running

- Install `go` with your package manager
- `go build whatever.go` will produce an executable called `whatever`
- After compiling, you can just run `./whatever`
  - No `CLASSPATH` stuff, `JAVA_HOME`, etc.!
- Or, you can do both in a single step:

  `go run whatever.go`
  - Note: an executable will not be produced in this case

Continuing our Tour

Let’s look at a few basic things that we expect all languages to have…

Variables

```
package main
// Use this format for multiple imports. Convention is to always do this,
// even if you only have a single import.
import (
    "fmt"
    "strconv"
)
func main() {
    var str = "Hello " + "world" + "!"     // We didn't specify a type!
    var magicNumber int = 42               // Notice how we gave this one a type
    f := 3.0                           // := is shorthand for 'var f = ...'
    var a, b, c = 1, 2, 3                  // Creating and assigning many ints

    fmt.Println(str + "  The magic number is: " + strconv.Itoa(magicNumber))
    fmt.Printf("We can printf! f = %f\n", f)
    fmt.Printf("a = %d, b = %d, c = %d\n", a, b, c)
}
```

Comments

- `//` and `/* ... */`, just like C and Java. 'Nuff said.
- What about Doxygen / Javadoc style comments?
  - Write a comment above whatever it is you're documenting. That's it! (No special `/**` or similar to identify document comments)
    - Generate documentation with `godoc`
  - There are no special identifiers, e.g., `@param`.
    - The idea here is your variables should be named clearly enough that these are not necessary

Types

- As you've seen, go uses *type inference* to figure out what the type is for each variable
  - We can be explicit about the type if we want
- Basic types:
  - `bool`
  - `string`, `rune`
  - `int`, `uint`
  - `byte`
  - `float32`, `float64`

Conditionals

```
a := 35
if a > 64 {
    fmt.Println("Bigger than 64!")
} else if a > 32 {
    fmt.Println("Bigger than 32!")
} else {
    fmt.Println("Not bigger than 32 or 64!")
}
```

- No parentheses needed
- No **ternary if** in the language (`something ? true : false`)

Loops

There is only one type of loop: `for`

```
// The syntax we're used to, just without parentheses:
for i := 0; i < 100; i++ {
    fmt.Println(i)
}

// This is more like a 'while' loop:
i := 0
for i < 100 {
    fmt.Println(i)
    i = i + 1
}

// "Forever" infinite loop (like while(true) { ... })
for {
    fmt.Println("loop")
    break // 'continue' is also supported in for loops!
}
```

Functions

Getting `func` -y

```
func doSomething() { // Takes no params, doesn't return anything
}

func isThisClassOverYet() (bool, bool) { // Returns two booleans
    return false, true
}

func addThree(numberOne int, numberTwo int, numberThree int) int {
    return numberOne + numberTwo + numberThree
}

// We can omit all but the last type if they are the same:
func addThree(numberOne, numberTwo, numberThree int) int {
    return numberOne + numberTwo + numberThree
}
```

Getting More Advanced

- So far we've looked at types, variables, conditionals, loops, and functions
- Apart from switching around the order of things and eliminating some non-essential functionality, everything is probably what we'd expect to see so far.
- So we'll look at the things that make Go a bit more unique… after you have a chance to actually write some code.

Today's Schedule

- Thoughts on AI
- Go Basics
- **A break (to write some Go)**

Break Time

1. Set up go on your development machine
2. Make sure you can log onto the CS network with `ssh`
   - `ssh USERNAME@stargate.cs.usfca.edu`
   - Default password is your student ID
   - Contact support@cs.usfca.edu if you have problems
3. Once you're there, you should be able to `ssh orionXX`
   - Where `XX` is a number from 01-12
   - We'll use this cluster for our projects

Lab 1

- Our first lab will be a crash course in go
- You'll write a log analyzer, something that actually *does* happen to be a big data problem 😉
- Head to the assignments page on the course website to create your repo!