Administrivia

Lab 1

- Lab 1 grading complete.
- Many students lost style points. Find out why at my office hours.
  - Main issue was not aligning style with existing code.
- Lab 2 being released tonight.
  - Use `gofmt` to avoid style point loss.

Go Programming Language

Today we’re talking about Go. Get feel for programming language experience:

* C++?
* Java?
* Python?

We’re going to start by looking at Rob Pike’s slides on Go.

Who is Rob Pike?

* Principal Engineer at Google, Inc.
* Works on distributed systems, data mining, programming languages, and software development tools.
* Before Google, was at Bell Labs.
* Worked on computer graphics, user interfaces, languages, concurrent programming, and distributed systems.
* Notable for Plan 9 (with Ken Thompson) and Inferno (Plan 9 successor)
* Co-author with Brian Kernighan of The Unix Programming Environment and The Practice of Programming.
* Wrote sam, acme. (text editors)

Who is Ken Thompson?

* We’ve seen this name before: Reflections on trusting trust.
* (early) Turing Award Winner!
* Design and implemented Unix (with late Dennis Ritchie, who did C)!
* Invented B programming language.
* Designed UTF-8 (with Rob Pike) most common encoding of Unicode code points.
* Helped design Go (with Rob Pike).
* Now works at Google.

Looked at Pike’s slides.

A tiny (in-lecture) tour of Go.

Types:

* bool
* string
* int int8 int16 int32 int64
* uint uint8 uint16 uint32 uint64 uintptr
* byte // alias for uint8
* rune // alias for int32, represents a Unicode code point
* float32 float64
* complex64 complex128

* arrays: [N]type (len)
* slices: []type (cap, len, append, range [i, value])
* map: map[keyType]valueType (range [key, val], delete)

- channels: chan type

Features:

- functions (‘func‘, return type (+ named), types after, compare with C)
- variables (var vs. := vs. =, multiple in one line)
- packages (see src/packages) (import rename, export with Caps)
- arrays, slices, maps (see tiny_tour)
  - arrays: x := [10]int{1, 2, 3, 4}
  - slice: x := []int{1, 2, 3, 4}
  - map: x := make(map[string]int)
- for (as C, while, forever, with range)
- if (with init (if thing := something; condition), error)
- [*] mention ‘switch‘:
  - with expression: switch ’expr’ { case val: ... }
- structs (var v struct { ... }, v.field = val), &struct, return pointer
- type (type struct, etc.)
- make vs. new (make(T, len, cap) is for "generics": slices, maps, channels)
- defer (see src/panic)
- error
- interfaces
- [*] panic/defer/recover (src/panic)

- goroutines, channels (buffered vs. unbuffered), close, select

----

Let’s try to reason about how to do a few things in Go that we’d do in other languages.

Four "design patterns":

1) Iterator: want a function/structure that can be iterated over. (src/iter)

  Q: How would we do this in Python?

```
def my_iter(end):
  for i in range(0, end):
    yield i

for value in my_iter(10):
  print value
```

  Q: How would we do this in Go?

```
func my_iter(end int) chan in {
    ch := make(chan int)
    go func() {
        for i := 0; i < end; i++ {
            ch <- i
        }
    }()

    return ch
}

func main() {
    for i := range my_iter(10) {
        fmt.Println(i)
    }
}
```

2) Singleton: want a single instance of a thing in the entire program.

  Q: How would we do this in Java?

```
public class ClassicSingleton {
  private static ClassicSingleton instance = null;
  protected ClassicSingleton() {
    // Exists only to defeat instantiation.
  }
  public static ClassicSingleton getInstance() {
    if (instance == null) {
      instance = new ClassicSingleton();
    }
    return instance;
  }
}
```

Q: Can we accomplish the same thing in Go?
A: Almost. Have package that exports an interface, has a private struct type implement it, and then has a function that return an instance of that struct as the Interface. Others can still implement the interface.

3) Adapter: have two interfaces, A and B, that are incompatible, but want to use an A as a B. (for example, function needs a B, want to use A as a B).

Q: How to do this in Java?
A: New class C that wraps A, inherit from B, implement methods.

Q: How do do this in Go?
A: Can just implement thing directly.

4) Template methods: Have some interface. Want default implementation of some of those things. (see src/template)

Q: How to do this in Java?
A: Have abstract class with default implementation and abstract methods. Inherit from it and overrideable methods.

Q: How to do this in Go?
A: Have interface with methods. Have "base" struct that implements the default methods. Have "children" structs embed base struct and implement the overrided methods.