Concurrent Programming

This lesson introduces the concept of concurrent programming

WE'LL COVER THE FOLLOWING

Introduction

Concurrent programming is a large topic, but it's also one of the most interesting aspects of the Go language.

Concurrent programming in many environments is made difficult by the subtleties required to implement correct access to *shared variables*. Go encourages a different approach in which shared values are passed around on channels and, in fact, never actively shared by separate threads of execution. Only one goroutine has access to the value at any given time.

Data races cannot occur, by design. To encourage this way of thinking we have reduced it to a slogan:

> Do not communicate by sharing memory; instead, share memory by communicating.

This approach can be taken too far. Reference counts may be best done by putting a *mutex* around an integer variable, for instance. But as a high-level approach, using channels to control access makes it easier to write clear, correct programs.

Although Go's approach to concurrency originates in Hoare's Communicating Sequential Processes (CSP), it can also be seen as a type-safe generalization of Unix pipes.

- Rob Pike's concurrency slides (IO 2012)

- Video of Rob Pike at IO 2012
- Video of Concurrency is not parallelism (Rob Pike)

In the next lesson, we'll introduce you to the interesting concept of *goroutines*, so keep on reading!