More On Go

1. Blockchain and Genetic Tracking - Blockchain could be used to track 2019-nCov
2. Namebase Uncensorable Web Domains go live.
3. Entering the “uncanny valley”.

Blockchain and Mining

What is Mining and How is it implemented.

1. More on Go

Maps do not synchronize automatically. So... Synchronization Primitives:

```
package main

import (
        "fmt"
        "sync"
        "time"
)

// SafeCounter is safe to use concurrently.
type SafeCounter struct {
        v   map[string]int
        mux sync.Mutex
}

// Inc increments the counter for the given key.
func (c *SafeCounter) Inc(key string) {
        c.mux.Lock()
        // Lock so only one goroutine at a time can access the map c.v.
        c.v[key]++
        c.mux.Unlock()
}

// Value returns the current value of the counter for the given key.
func (c *SafeCounter) Value(key string) int {
        c.mux.Lock()
        // Lock so only one goroutine at a time can access the map c.v.
        defer c.mux.Unlock()
        return c.v[key]
}
```

```
func main() {
        c := SafeCounter{v: make(map[string]int)}
        for i := 0; i < 1000; i++ {
                go c.Inc("somekey")
        }

        time.Sleep(time.Second)
        fmt.Println(c.Value("somekey"))
}
```

A Go Core/Panic

First the Code

```
        package main

        import "fmt"

        var mm map[string]int

        func main() {
                fmt.Println("vim-go")
                mm["bob"] = 3
        }
```

Then the bad output.

```
panic: assignment to entry in nil map

goroutine 1 [running]:
panic(0x10a5540, 0x10d03a0)
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/panic.go:551 +0x3c1 fp=0xc42005
runtime.mapassign_faststr(0x10a4e80, 0x0, 0x10be68a, 0x3, 0x0)
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/hashmap_fast.go:696 +0x407 fp=0
main.main()
        /Users/corwin/go/src/github.com/Univ-Wyo-Education/Blockchain-4010-Fall-2018/Le
runtime.main()
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/proc.go:198 +0x212 fp=0xc42005f
runtime.goexit()
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/asm_amd64.s:2361 +0x1 fp=0xc420

goroutine 2 [force gc (idle)]:
runtime.gopark(0x10c5580, 0x11397a0, 0x10bfafe, 0xf, 0x10c5414, 0x1)
```

```
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/proc.go:291 +0x11a fp=0xc42004a
runtime.goparkunlock(0x11397a0, 0x10bfafe, 0xf, 0x14, 0x1)
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/proc.go:297 +0x5e fp=0xc42004a7
runtime.forcegchelper()
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/proc.go:248 +0xcc fp=0xc42004a7
runtime.goexit()
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/asm_amd64.s:2361 +0x1 fp=0xc420
created by runtime.init.4
        /usr/local/Cellar/go/1.10.3/libexec/src/runtime/proc.go:237 +0x35
```

Pseudo Code for Mining (Homework 02)

```
package mine

import "github.com/Univ-Wyo-Education/S19-4010/a/02/block"

// TODO Replace above import with import below (commented out)
/*
import (
        "encoding/hex"
        "fmt"

        "github.com/Univ-Wyo-Education/S19-4010/a/02/block"
        "github.com/Univ-Wyo-Education/S19-4010/a/02/hash"
)
*/

// MineBlock implements a proof of work mining system where the first 4 digits (2 bytes
// Difficulty can be increaesed by requiring more digits to be 0 or by requring some ot
// the resulting hash.
func MineBlock(bk *block.BlockType, difficulty string) {
        // Pseudo-Code
        //
        // 1. Use an infinite loop to:
        //   1. Serialize the data from the block for hashing, Call `block.SearilizeFor
        //   2. Calculate the hash of the data, Call `hash.HashOf` to do this. This is
        //      replaced the software with a hash calculator on a graphics card where y
        //      What would happen if we replaced the graphics card with an ASIC - so yo
        //      the hash and you could run 4 billion hashes a second?
        //   3. Convert the hash (it is []byte) to a hex string.  Use the `hex.EncodeTo
        //   4. `fmt.Printf("((Mining)) Hash for Block [%s] nonce [%8d]\r", theHashAsAS
        //                      `\r` will overwrite the same line instead of advancing
        //   5. See if the first 4 characters of the hash are 0's. - if so we have met
        //      In go this is `if theHashAsAString[0:4] == "0000" {`.  This is create a
        //      character 0 with length of 4, then compare that to the string `"0000"`.
        //    - Set the block's "Seal" to the hash
        //    - `fmt.Printf("((Mining)) Hash for Block [%s] nonce [%8d]\n", theHashAsAS
```

```
        //                      `\n` will overwrite the same *and then advance* to th
        //    - return
        //   5. Increment the Nonce in the block, and...
        //   6. Back to the top of the loop for another try at finding a seal for this
        //
        // For the genesis block, when I do this it requires 54586 trips through the lo
        // proof of work.
        //

        // TODO: Start coding here.
}
```