Lecture 5 - More on Mining, Go Interfaces and Go Weaknesses

News

1. Hack-A-Thon
2. Hack-A-Thon
3. China is accepting digital digests (hash) as evidence in a court of law.
4. Australia is putting drivers licenses on the blockchain

Detailed walk through of mining.

Walk Through

```
1. Use an infinite loop to:
  1. Serialize the data from the block for hashing, Call `block.SerializeForSeal` to do
  2. Calculate the hash of the data, Call `hash.HashOf` to do this. This is the slow pa
     replaced the software with a hash calculator on a graphics card where you could ru
     What would happen if we replaced the graphics card with an ASIC - so you had dedic
     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.EncodeToString` sta
  4. `fmt.Printf("((Mining)) Hash for Block [%s] nonce [%8d]\r", theHashAsAString, bk.N
  5. See if the first 4 characters of the hash are 0's. - if so we have met the work cr
     In go this is `if theHashAsAString[0:4] == "0000" {`.  This is create a slice, 4 l
     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", theHashAsAString, bk.N
   - 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 block.
```

Go Interfaces

Two uses for interfaces (Actually more than 2 but 2 primary uses).

1. Variable parameter list functions.
2. Interfaces to sets of functions.

Variable parameter list functions.

```
func vexample(a int, b ...interface{}) {
        for pos, bVal := range b {
                switch v := bVal.(type) {
                case int:
                        fmt.Printf("It's an int, %d at %d\n", v, pos)
                case []int:
                        fmt.Printf("It's a slice of int\n")
                default:
                        fmt.Printf("It's a something else\n")
                }
        }
}
```

Interfaces to sets of functions.

```
type InterfaceSpecType interface {
        DoFirstThing(p1 int, p2 int) error
        DoSomethingElse() error
}

type ImplementationType struct {
        AA int
        BB int
}

var _ InterfaceSpecType = (*ImplementationType)(nil)


func NewImplementationType() InterfaceSpecType {
        return &ImplementationType{
                AA: 1,
                BB: 2,
        }
}

func (xy *ImplementationType) DoFirstThing(p1 int, p2 int) error {
        // ... do something ...
        return nil
}

func (xy *ImplementationType) DoSomethingElse() error {
        // ... do something ...
        return nil
}

func Demo() {
        var dd InterfaceSpecType
        dd = NewImplementationType()
        _ = dd.DoSomethingElse()
}
```

Go Weaknesses

What are the limitations of using Go

1. No objects - Use interfaces instead. No inheritance.
2. No generics - Use templates and code instead.
3. No error handling - Just return errors.

Go 2.0 is coming in 1.5 years. Go's design team commitment is 100% backward compatibility - it will be able to correctly compile go 1.0 code without change to the language.