**Module 1 – 1 RFCs**

1. Protocols and Formats

* Requests for Comments (RFC)
* Definitions of internet protocols and formats
* Example protocols : HTML – Hypertext Markup Language, 1866 and URI – Uniform Resource Identifier, 3986 and HTTP – Hypertext Transfer Protocols, 2616

1. Protocol Packages

* Golang provides packages for important RFCs
* Functions which encode and decode protocol format
* Example:
  + “net/http”
  + Web communication protocol
  + http.Get([www.uci.edu](http://www.uci.edu))
  + “net”
  + TCP/IP and socket programming
  + net.Dial(“tcp”, “uci.edu.80”)

1. JSON

JavaScript Object Notation

RFC 7159

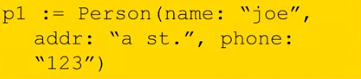
Format to represent structured information

Attribute-value pairs: struct or map

Basic value types: bool, number, string, array, “object”

1. JSON Example

* Go struct



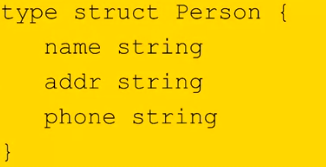
* Equivalent JSON object

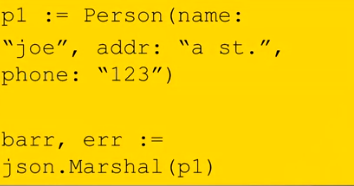


**Module 1 – 2 JSON**

1. JSON Properties

* All Unicode
* Human readable
* Fairly compact representation
* Types can be combined recursively: Array of structs, struct in struct
* Generating JSON representation from an object





* Marshal() returns JSON representation as [] byte

1. JSON Unmarshalling

* Unmarshal() converts a JSON [] byte into a Go object



* Pointer to Go object is passed to Unmarshal()
* Object must “fit” JSON [] byte

**Module 2 – 1 File Access, ioutil**

1. Files

* Linear access, not random access: mechanical delay
* Basic operations:
* Open – get handle for access
* Read – read bytes into [] byte
* Write – write [] byte into file
* Close – release handle
* Seek – move read / write head

1. Ioutil File Read

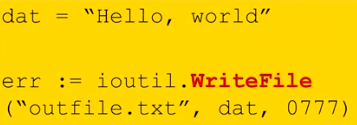
* “io/ioutil” package has basic functions



* dat is [] byte filled with contents of entire file
* Explicit open / close are not needed
* Large files cause a problem

1. Ioutil File Write

* Writes [] byte to file
* Creates a file
* Unix-style permission bytes



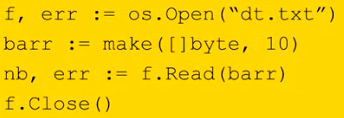
**Module 2 – 2 File Access, os**

1. os Package File Access

* os.open() opens a file, returns a file descriptor (File)
* os.close() closes a file
* os.read() reads from a file into a [] byte, fills the [] byte and control the amount read
* os.write() writes a [] byte into a file

1. os File Reading

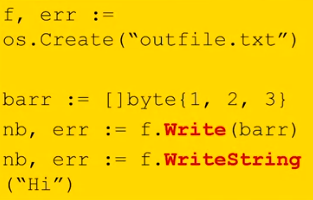
* Opening and Reading



* Reads and fills barr
* Read returns # of bytes read
* May be less than [] byte length

1. os File Create / Write

* WriteString() writes a string



* Write() writes a [] byte: any unicode sequence