**HTTP server using Go:**

A fundamental concept in net/http servers is handlers. A handler is an object implementing the http.Handler interface.

A common way to write a handler is by using the http.HandlerFunc adapter on functions with the appropriate signature.

Functions serving as handlers take a http.ResponseWriter and a http.Request as arguments. The response writer is used to fill in the HTTP response.

We register our handlers on server routes using the http.HandleFunc convenience function. It sets up the default router in the net/http package and takes a function as an argument.

Finally, we call the ListenAndServe with the port and a handler. nil tells it to use the default router we’ve just set up.

**Example program:**

package main

import (

"fmt"

"net/http"

)

func hello(w http.ResponseWriter, req \*http.Request) {

fmt.Fprintf(w, "hello\n")

}

func headers(w http.ResponseWriter, req \*http.Request) {

for name, headers := range req.Header {

for \_, h := range headers {

fmt.Fprintf(w, "%v: %v\n", name, h)

}

}

}

func main() {

http.HandleFunc("/hello", hello)

http.HandleFunc("/headers", headers)

http.ListenAndServe(":8090", nil)

}

**Output:**

Hello

----------------------------------------------------------------------------------------------------

**Json Marshalling and UnMarshalling using go:**

JSON is a widely used format for data interchange. Golang provides multiple encoding and decoding APIs to work with JSON including to and from built-in and custom data types using the encoding/json package.

**Data Types**: The default Golang data types for decoding and encoding JSON are as follows:

* bool for JSON booleans
* float64 for JSON numbers
* string for JSON strings
* nil for JSON null
* array as JSON array
* map or struct as JSON Object

**Encoding/Marshaling structs:**

The Marshal() function in package encoding/json is used to encode the data into JSON.

**Syntax:**

func Marshal(v interface{}) ([]byte, error)

**Program:**

package main

import (

"fmt"

"encoding/json"

)

type Human struct{

Name string

Age int

Address string

}

func main() {

// defining a struct instance

human1 := Human{"Ankit", 23, "New Delhi"}

// encoding human1 struct

// into json format

human\_enc, err := json.Marshal(human1)

if err != nil {

fmt.Println(err)

}

// as human\_enc is in a byte array

// format, it needs to be

// converted into a string

fmt.Println(string(human\_enc))

human2 := []Human{

{Name: "Rahul", Age: 23, Address: "New Delhi"},

{Name: "Priyanshi", Age: 20, Address: "Pune"},

{Name: "Shivam", Age: 24, Address: "Bangalore"},

}

human2\_enc, err := json.Marshal(human2)

if err != nil {

fmt.Println(err)

}

fmt.Println()

fmt.Println(string(human2\_enc))

}

**Output:**

{"Name": "Ankit", "Age":23, "Address": "New Delhi"}

[{"Name": "Rahul", "Age":23, "Address": "New Delhi"}, {"Name": "Priyanshi", "Age":20, "Address": "Pune"}, {"Name": "Shivam" , "Age":24, "Address": "Bangalore"}]

**Decoding/Unmarshaling structs:**

The Unmarshal() function in package encoding/json is used to unpack or decode the data from JSON to struct.

**Syntax:**

func Unmarshal(data []byte, v interface{}) error

**Program:**

package main

import (

"fmt"

"encoding/json"

)

type Human struct{

// defining struct variables

Name string

Address string

Age int

}

func main() {

var human1 Human

// data in JSON format which

// is to be decoded

Data := []byte(`{

"Name": "Deeksha",

"Address": "Hyderabad",

"Age": 21

}`)

// decoding human1 struct

// from json format

err := json.Unmarshal(Data, &human1)

if err != nil {

// if error is not nil

// print error

fmt.Println(err)

}

// printing details of

// decoded data

fmt.Println("Struct is:", human1)

fmt.Printf("%s lives in %s.\n", human1.Name, human1.Address)

// unmarshaling a JSON array

// to array type in Golang

// defining an array instance

// of struct type

var human2 []Human

// JSON array to be decoded

// to an array

Data2 := []byte(`

[

{"Name": "Vani", "Address": "Delhi", "Age": 21},

{"Name": "Rashi", "Address": "Noida", "Age": 24},

{"Name": "Rohit", "Address": "Pune", "Age": 25}

]`)

// decoding JSON array to

// human2 array

err2 := json.Unmarshal(Data2, &human2)

if err2 != nil {

// if error is not nil

// print error

fmt.Println(err2)

}

**Output:**

Struct is: {Deeksha Hyderabad 21}

Deeksha lives in Hyderabad.

{Vani Delhi 21}

{Rashi Noida 24}

{Rohit Pune 25}

// printing decoded array

// values one by one

for i := range human2{

fmt.Println(human2[i])

}

}