Go-Basics

I start from testing theoretical part.

Theoretical part

- **Variables**

```
// variables go

   var t = 1

   fmt.Println(t)

   var a, c int = 2, 4

   fmt.Println(a + c)

   var d = true

   fmt.Println(d)

   var e int

   fmt.Println(e)

   f := "hey"

   fmt.Println(f)
```

- **Pointers**

```
// pointer

   i, j := 20, 400

   p := &i

   fmt.Println("our p :", *p)

   *p = 40
   fmt.Println("now i is equal 40: ", i)

   p = &j

   *p = *p / 2

   fmt.Println("our p have value j and divide 2 :", j)
```

- **Functions**

```
//function

func add(i int, j int) int {

   //condition if

   if i+j > 200 {

       var a = i - j

       fmt.Println("Condition if is working:", a)

   }

   return i + j

}

// functions

   fmt.Println("Show result from function : ", add(i, j))
```

- **Conditional: if / switch …**

```
//condition if

   if i+j > 200 {

       var a = i - j

       fmt.Println("Condition if is working:", a)
   }
```

- **Loops for**

```
// loop for

   sum := 0

   for i := 0; i <= 10; i++ {

       sum = i

   }

   fmt.Println("Result from loop:", sum)
```

- **Arrays / Slices**

```
//array

   var arr [2]int

   arr[0] = 1

   arr[1] = 2

   fmt.Println("number of arr[0]", arr[0])

   fmt.Println("numbers in array: ", arr)
```

```
//slices

   numbers := [4]int{1, 2, 3, 4}

   var q []int = numbers[0:4]
   fmt.Println("slices numbers : ", q)
```

- **Maps**

```
//maps

   type Vertex struct {

       Lat, Long float64

   }

   var n map[string]Vertex

   n = make(map[string]Vertex)

   n["bell labs"] = Vertex{

       40.40342, -10.3242,

   }

   fmt.Println("Maps: ", n["bell labs"])
```

- **Structs and user-defined types**

```
// structs is a coolection of fileds

   type Test struct {

       X int

       Y int

   }

   fmt.Println("Struct test : ", Test{1, 2})

   // definde types struct

   type Car struct {
       name   string

       year   int

       speed int

   }

   var model Car

   model.name = "mini"

   model.year = 2000

   model.speed = 140

   fmt.Println("Type struct:", model)
```

- **Interfaces**

```
//interface

type animal interface {

   eat()

   showAge()

}

type dog struct {

   name string

   age  int

}

func (d dog) showAge() {

   fmt.Println("The dog", d.name, "have", d.age, "years.")

}
```

```
func (d dog) eat() {

   fmt.Println("dog eat", d.name)

}

func test(a animal) {

   fmt.Println(a)

}

//interface and polymorphism

   fmt.Println("Interface:")

   d1 := dog{"Felix", 34}

   d2 := dog{"Dolly", 12}

   d1.showAge()

   d1.eat()

   d2.eat()

   test(d1)

   test(d2)
```

- **Errors**

```
img, err := os.Open("scooby.jpg")
   if err != nil {
       log.Fatal(err)
   }
```

Practical part

Task list :

1. Create server
2. Check server if is working with status code 200
3. Create HTML template
4. Create routing
5. Create response for main page with html template
6. Create response for request in XML
7. Create response for request in JSON
8. Create response for request in plain text
9. Create response for request in image

Priority of each task is the same like numbers in task list i start from 1 what is Create server.

Create server

import

```
   "net/http"
```

func main()

```
   log.Fatal(http.ListenAndServe(":3000", nil))
```

Check server status code 200

```
w.Header().Set("Content-Type", "text/html; charset=utf8")
   w.WriteHeader(http.StatusOK)
```

Create HTML template

template.html

Create routing

```
func main() {
   http.HandleFunc("/", htmlHandler)
   http.HandleFunc("/xml", xmlHandler)
   http.HandleFunc("/json", jsonHandler)
   http.HandleFunc("/plain", plainHandler)
   http.HandleFunc("/image", imageHandler)
   log.Fatal(http.ListenAndServe(":3000", nil))

}
```

Create response for main page with html template

```
func htmlHandler(w http.ResponseWriter, r *http.Request) {
   w.Header().Set("Content-Type", "text/html; charset=utf8")
   w.WriteHeader(http.StatusOK)
   t, e := template.ParseFiles("template.html")
   t.Execute(w, e)
}
```

Create response for request in XML

```
func xmlHandler(w http.ResponseWriter, r *http.Request) {

   profile := Profile{"Wojtek", []string{"Travels",
"Programming"}}

   x, err := xml.MarshalIndent(profile, "", "")
   if err != nil {
       http.Error(w, err.Error(), http.StatusInternalServerError)
       return
   }
   w.Header().Set("Content-type", "text/xml; charset=utf8")
   w.WriteHeader(http.StatusOK)
   w.Write(x)
}
```

Create response for request in JSON

```
func jsonHandler(w http.ResponseWriter, r *http.Request) {
   w.Header().Set("Content-type", "application/json;
charset=utf8")
   w.WriteHeader(http.StatusOK)
   profile := Profile{"Wojtek", []string{"Travels",
"Programming"}}
   js, err := json.MarshalIndent(profile, "", " ")
   if err != nil {
       http.Error(w, err.Error(), http.StatusInternalServerError)
       return
   }
```

```go
    w.Write(js)
}
```

Create response for request in plain text

```go
func plainHandler(w http.ResponseWriter, r *http.Request) {
    profile := Profile{"Wojtek", []string{"Travels",
"Programming"}}
    plain, err := json.MarshalIndent(profile, "", " ")
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    w.Header().Set("Content-type", "text/plain; charset=utf8")
    w.WriteHeader(http.StatusOK)
    w.Write(plain)
}
```

Create response for request in image

```go
func imageHandler(w http.ResponseWriter, r *http.Request) {
    img, err := os.Open("scooby.jpg")
    if err != nil {
        log.Fatal(err)
    }
    defer img.Close()
    w.Header().Set("Content-type", "image/jpeg; charset=utf8")
    w.WriteHeader(http.StatusOK)
    io.Copy(w, img)
}
```