Golang Coding Bootcamp 2021

Lecture 01

- Data Type
  - character = A, B, c, d..
  - string = "I am a student"
  - Integar = int = 1, 2, 3..
  - float = 10.25
  - date = 06-01-2023
  - time = 12:13:16
  - date-time = 06-01-2023 13:12:25

Lecture 02

- Course Outline Link
  - https://docs.google.com/document/d/1ngy6N91ZgGxWSk8RepqnzMU-uk0dukBX3qV0VUTBS_o
- How Computer Works
  - Main 4 Steps
    - Input
      - Mouse
      - Keyboard
      - Scanner
    - Store
      - Volatile = Non Permanent Memory
        - RAM = Random Access Memory
      - Non Volatile = Non Changable
        - Hard Drive
        - Solid State Drive
    - Process
      - Processor/ Central Processing Unit
    - Output
      - Monitor
      - Soundbox
- ASCII
  - ASCII (American Standard Code for Information Interchange) is the most common character encoding format for text data in computers and on the internet. In standard ASCII-encoded data, there are unique values for 128 alphabetic, numeric or special additional characters and control codes.
- Decimal to Binary Conversion
  - https://www.youtube.com/watch?v=2U9b76JRz7s

Lecture 03

- What is Computer Programming?
  - Computer programming is the process of writing code to facilitate specific actions in a computer, application or software program, and instructs them on how to perform. Computer programmers are professionals that create instructions for a computer to execute by writing and testing code that enables applications and software programs to operate successfully.
    - Design & Building
    - Executable Computer Program
    - Perform Specific Task & Gives expected result
- Golang
  - https://go.dev/dl/go1.20.windows-amd64.msi
    - Download and Install

Lecture 04

- Basic Commands
  - Go to Command Prompt
    - type "e:" & enter to go to E:/ drive
    - type "dir" & enter to files and directory from command prompt
    - type "mkdir folder_name" to create folder
    - type "echo hello > student.txt" to create new file
    - type "cd .." to go to pervious drive folder
    - For More : https://www.youtube.com/playlist?list=PL6gx4Cwl9DGDV6SnbINlVUd0o2xT4JbMu
    - type "set" to get env variable from pc
- GOPATH
  - GOPATH is a variable that defines the root of your workspace. By default, the workspace directory is a directory that is named go within your user home directory (~/go for Linux and MacOS, %USERPROFILE%/go for Windows). GOPATH stores your code base and all the files that are necessary for your development.
    - in cmd prompt type "setx GOPATH "D:/GO_PROJECT"" to set go path
- GOROOT
  - GOROOT is a variable that defines where your Go SDK is located. You do not need to change this variable, unless you plan to use different Go versions. GOPATH is a variable that defines the root of your workspace.
- GO project folder structure
  - bin — All executable binary
  - pkg — packages/library
  - src — source code

Lecture 05

- Tasks, List of Tasks, Sequence
- File: A file is a container in a computer system for storing information. Files used in computers are similar in features to that of paper documents used in library and office files.
  - Folder: A folder is a group of files that are stored together on a computer.
    - Package: Go programs are organized into packages. A package is a collection of source files in the same directory that are compiled together. Functions, types, variables, and constants defined in one source file are visible to all other source files within the same package.
      - Extention: A file extension, or filename extension, is a suffix at the end of a computer file. It comes after the period and is usually two to four characters long. If you've ever opened a document or viewed a picture, you've probably noticed these letters at the end of your file.

        File extensions are used by the operating system to identify what apps are associated with what file types—in other words, what app opens when you double-click the file. For example, a file named "awesome_picture.jpg" has the "jpg" file extension. When you open that file in Windows, for example, the operating system looks for whatever app is associated with JPG files, opens that app, and loads the file.
- First Program

```
package main

import (
 "fmt"
 "runtime"
)

func main() {

 fmt.Println("Hello World, This is my first go program")

 fmt.Printf("Hello, version of t his go is %s\n", runtime.Version())

}
```

  - to run "go main.go"

Lecture 06,07

- Install git

SETUP
Configuring user information used across all local repositories

git config --global user.name "[firstname lastname]"
set a name that is identifiable for credit when review version history

git config --global user.email "[valid-email]"
set an email address that will be associated with each history marker

git config --global color.ui auto
set automatic command line coloring for Git for easy reviewing

SETUP & INIT
Configuring user information, initializing and cloning repositories

git init
initialize an existing directory as a Git repository

git clone [url]
retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT
Working with snapshots and the Git staging area

git status
show modified files in working directory, staged for your next commit

git add [file]
add a file as it looks now to your next commit (stage)

git reset [file]
unstage a file while retaining the changes in working directory

git diff
diff of what is changed but not staged

git diff --staged
diff of what is staged but not yet committed

git commit -m "[descriptive message]"
commit your staged content as a new commit snapshot

BRANCH & MERGE
Isolating work in branches, changing context, and integrating changes

git branch
list your branches. a * will appear next to the currently active branch

git branch [branch-name]
create a new branch at the current commit

git checkout
switch to another branch and check it out into your working directory

git merge [branch]
merge the specified branch's history into the current one

git log
show all commits in the current branch's history

IGNORING PATTERNS
Preventing unintentional staging or commiting of files

git config --global core.excludesfile [file]
system wide ignore pattern for all local repositories

SHARE & UPDATE
Retrieving updates from another repository and updating local repos

git remote add [alias] [url]
add a git URL as an alias

git fetch [alias]
fetch down all the branches from that Git remote

git merge [alias]/[branch]
merge a remote branch into your current branch to bring it up to date

git push [alias] [branch]
Transmit local branch commits to the remote repository branch

git pull
fetch and merge any commits from the tracking remote branch

Lecture 08

- Syntax is a set of rules

```
package main
import ("fmt")

func main() {
  fmt.Println("Hello World!")
}
```

Example explained
Line 1: In Go, every program is part of a package. We define this using the package keyword. In this example, the program belongs to the main package.

Line 2: import ("fmt") lets us import files included in the fmt package.

Line 3: A blank line. Go ignores white space. Having white spaces in code makes it more readable.

Line 4: func main() {} is a function. Any code inside its curly brackets {} will be executed.

Line 5: fmt.Println() is a function made available from the fmt package. It is used to output/print text. In our example it will output "Hello World!".

Go Statements
fmt.Println("Hello World!") is a statement.

In Go, statements are separated by ending a line (hitting the Enter key) or by a semicolon ";".

Hitting the Enter key adds ";" to the end of the line implicitly (does not show up in the source code).

The left curly bracket { cannot come at the start of a line.

Run the following code and see what happens:

- variables

```
package variables

import "fmt"

func Variables() {

    var a = "initial"
    fmt.Println(a)

    var b, c int = 1, 2
    fmt.Println(b, c)

    var d = true
    fmt.Println(d)

    var e int
    fmt.Println(e)

    f := "apple"
    fmt.Println(f)
}
```

- Cheat Sheets: https://cheatography.com/explore/search/?q=golang

Lecture 09

- Data Types

Go is statically typed, meaning that once a variable type is defined, it can only store data of that type.

Go has 3 basic data types:

bool: represents a boolean value and is either true or false
Numeric: represents integer types, floating point values, and complex types
string: represents a string value

```
package main
import ("fmt")

func main() {
  var a bool = true     // Boolean
  var b int = 5         // Integer
  var c float32 = 3.14  // Floating point number
  var d string = "Hi!"  // String
  var chr rune = "h" //rune

  fmt.Println("Boolean: ", a)
  fmt.Println("Integer: ", b)
  fmt.Println("Float:   ", c)
  fmt.Println("String:  ", d)
  fmt.Println("Rune:  ", chr)
  fmt.Printf("Char:  %c", chr)
}
```

- Taking Input From User

```
package main
import ("fmt")

func main() {
  var i int

/*
The Scan() function receives the user input in raw format as space-separated values and stores them in the variables. Newlines count as spaces.
 */
  fmt.Print("Type a number: ")
  fmt.Scan(&i)
  fmt.Println("Your number is:", i)

/*
The Scanln() function is similar to Scan(), but it stops scanning for inputs at a newline (at the press of the Enter key.).
*/
  var i,j int

  fmt.Print("Type two numbers: ")
  fmt.Scanln(&i, &j)
  fmt.Println("Your numbers are:", i, "and", j)

/*
The Scanf() function receives the inputs and stores them based on the determined formats for its arguments.
*/
  var i,j int

  fmt.Print("Type two numbers: ")
  fmt.Scanf("%v %v",&i, &j)
  fmt.Println("Your numbers are:", i, "and", j)
}
```

Lecture 10

- Array

```
package main

import "fmt"

// In Go, an array is a numbered sequence of elements of a specific length.

func main() {

 var a [5]int
    fmt.Println("emp:", a)

    a[4] = 100
    fmt.Println("set:", a)
    fmt.Println("get:", a[4])

    fmt.Println("len:", len(a))

    b := [5]int{1, 2, 3, 4, 5}
    fmt.Println("dcl:", b)

    c := [...]int{1, 2, 3, 4, 5}
    fmt.Println("dcl:", c)
}
```

- Slice

```
package main

import "fmt"

/*
In Go language slice is more powerful, flexible, convenient than an array,
and is a lightweight data structure. Slice is a variable-length sequence which stores elements of a similar type,
we are not allowed to store different type of elements in the same slice. It is just like an array having an index value and length,
but the size of the slice is resized they are not in fixed-size just like an array.
Internally, slice and an array are connected with each other, a slice is a reference to an underlying array. It is allowed to store duplicate elements in the slice.
The first index position in a slice is always 0 and the last one will be (length of slice – 1).
*/

func main() {

    s := make([]string, 3)
    fmt.Println("emp:", s)

    s[0] = "a"
    s[1] = "b"
    s[2] = "c"
    fmt.Println("set:", s)
    fmt.Println("get:", s[2])

    fmt.Println("len:", len(s))

    s = append(s, "d")
    s = append(s, "e", "f")
    fmt.Println("apd:", s)

    c := make([]string, len(s))
    copy(c, s)
    fmt.Println("cpy:", c)

    l := s[2:5]
    fmt.Println("sl1:", l)

    l = s[:5]
    fmt.Println("sl2:", l)

    l = s[2:]
    fmt.Println("sl3:", l)

    t := []string{"g", "h", "i"}
    fmt.Println("dcl:", t)
}
```

- Map

```
package main

import "fmt"

// Maps are Go's built-in associative data type (sometimes called hashes or dicts in other languages).

func main() {

    m := make(map[string]int)

    m["key_1"] = 7
    m["key_2"] = 13

    fmt.Println("map:", m)

    value_1 := m["key_1"]
    fmt.Println("value_1: ", value_1)

    fmt.Println("len:", len(m))

    delete(m, "key_2")
    fmt.Println("map:", m)
}
```

Lecture 11

- Struct

```
package main

import "fmt"

type person struct {
 name string
 age int
}

func newPerson(name string, age int) *person {
 p := person{name: name, age: age}
 return &p
}

func main() {
 fmt.Println(person{"Bob", 20})
 fmt.Println(person{name:"Bob", age:20})
 fmt.Println(person{name:"Bob"})
 fmt.Println(&person{name:"Bob", age:30})
 fmt.Println(newPerson("Bob",30))

 s := person{name:"Bob", age:210}
 fmt.Println(s.age)

 sp := &s
 sp.age = 12
 fmt.Println(sp.age, s.age)
}
```

- Variable Scope
  - to mutiple file run "go ."

```
// Program to illustrate global variable

package main
import "fmt"
/*
When we declare variables before the main() function, these variables will have global scope. We can access them from any part of the program.

These types of variables are called global variables. For example,
*/
// declare global variable before main function
var sum int

func addNumbers () {

  // local variable
  sum = 9 + 5
}


func main() {

  addNumbers()

  // can access sum
  fmt.Println("Sum is", sum)

}
```

Lecture 12

- For Loop

```
package main

import "fmt"

func main() {

 // Single condition
    i := 1
    for i <= 3 {
        fmt.Println(i)
        i = i + 1
    }

 // Classical For
    for j := 7; j <= 9; j++ {
        fmt.Println(j)
    }

 // Without a condition
    for {
        fmt.Println("loop")
        break
    }

 // With Continue
    for n := 0; n <= 5; n++ {
        if n%2 == 0 {
            continue
        }
        fmt.Println(n)
    }
}
```

- Conditional Statements
  - IF ELSE

```
package main

import "fmt"

func main() {

    if 12%4 == 0 {
        fmt.Println("12 is divisible by 4")
    }

    if 11%2 == 0 {
        fmt.Println("11 is even")
    } else {
        fmt.Println("11 is odd")
    }

    if num := 18; num < 0 {
        fmt.Println(num, "is negative")
    } else if num < 10 {
        fmt.Println(num, "has 1 digit")
    } else {
        fmt.Println(num, "has multiple digits")
    }

 /*
 There is no ternary if in Go, so you'll need to use a full
 if statement even for basic conditions.
 */
}
```

  - Switch Condition

```
package main

import (
    "fmt"
    "time"
)

func BasicSwitch() {

    i := 2
    fmt.Print("Write ", i, " as ")
    switch i {
    case 1:
        fmt.Println("one")
    case 2:
        fmt.Println("two")
    case 3:
        fmt.Println("three")
    }
}

func MultipleSwitch() {

 switch time.Now().Weekday() {
    case time.Saturday, time.Sunday:
        fmt.Println("It's the weekend")
    default:
        fmt.Println("It's a weekday")
    }
}
func main() {
 BasicSwitch()
 MultipleSwitch()
}
```

Lecture 13, 14

- Function

```
package main

import "fmt"

func plus(a int, b int) int {

    return a + b
}

func plusPlus(a, b, c int) int {
    return a + b + c
}

func main() {

    res := plus(1, 2)
    fmt.Println("1+2 =", res)

    res = plusPlus(1, 2, 3)
    fmt.Println("1+2+3 =", res)
}
```

- Pointers

```
package main

import "fmt"

func zeroval(ival int) {
 ival = 0
}

func zeroptr(iptr *int) {
 *iptr = 0
}

func main() {
 i := 1
 fmt.Println("initial:", i)
 fmt.Println("initial-pointer:", &i)

 zeroval(i)
 fmt.Println("zeroval:", i)
 fmt.Println("zeroval-pointer:", &i)

 zeroptr(&i)
 fmt.Println("zeroptr-value:", i)
 fmt.Println("zeroptr-pointer:", &i)
}
```

- Dereferencing Pointers

```
package main

import (
    "fmt"
)

type Person struct {
    Name string
}

func main() {
    c := new(Person) // returns pointer
    c.Name = "Catherine"
    fmt.Println(c.Name) // prints: Catherine
    d := c
    d.Name = "Daniel"
    fmt.Println(c.Name) // prints: Daniel
    // Adding an Asterix before a pointer dereferences the pointer
    i := *d
    i.Name = "Ines"
    fmt.Println(c.Name) // prints: Daniel
    fmt.Println(d.Name) // prints: Daniel
    fmt.Println(i.Name) // prints: Ines
}
```

Lecture 15

- OSI Model
  - https://www.geeksforgeeks.org/layers-of-osi-model/
- TCP/IP Model
  - https://www.geeksforgeeks.org/tcp-ip-model/