Lab Intro

Assignments

Project

Lab schedule

Bonus / Exam / Grading

- Assignments will have 1 week duration, late submission after the one week (10 -25 % penalty) of the total mark
- There will be one project (max 5 members) for the course this project needs to be joined with **Trello / clickup** as planning and team management and git repository **github / gitlab / bitbucket ...**
- Mon 2:30 - 5:30 LT / Wen 7:30 - 10:30 LT / Sat 3:00 - 6:00 LT
- There will be Bonuses throughout the class
- There might be a Lab Exam
- Grading criteria for the assignment and project will be set in respect to the task

Web Programing |

Lab

Introduction to Go (goLang)

Content

- Introduction
- Getting Started With Go
- Hello World
- Basics of the language
  - Variables
  - Functions
  - Flow Control statements (for... if.. Switch… defer)
- Next class
  - Structs , slice , maps
  - Methods and interface
  - concurrency

Next class

- Installing and using git
- Using git and gitlab
- Using clickup / trello

Introduction to Go (Golang)

Created by small team in google for getting the best of the main 3 languages which was used frequently at google then [python , java , c++]

It is an open source project

Features

- Strong and Statically typed (Java / c++)
- Simplicity
- Fast Compile times
- Build-in Concurrency
- Compile to Standalone Binaries

Getting started with Go

Official site

https://golang.org/dl/

Getting started and installation

https://golang.org/doc/install

Playground

https://play.golang.org/

Important References

How to Write Go Code

https://golang.org/doc/code.html#Workspaces

Go Tour

https://tour.golang.org/

Effective Go

https://golang.org/doc/effective_go.html

Downloading Go

Windows

https://dl.google.com/go/go1.13.1.windows-amd64.msi

Linux

https://dl.google.com/go/go1.13.1.linux-amd64.tar.gz

Apple MacOS

https://dl.google.com/go/go1.13.1.darwin-amd64.pkg

Go to the following page to download the latest version for your machine

https://golang.org/dl/

Installing Go

Linux, macOS

Extract the downloaded archive to **`/usr/local`** using the following command

```
tar -C /usr/local -xzf
go$VERSION.$OS-$ARCH.tar.gz
```

Windows

The msi installer do the configuration automatically

For other installation alternatives go to the following link

https://golang.org/doc/install

Go Editor for Your Platform

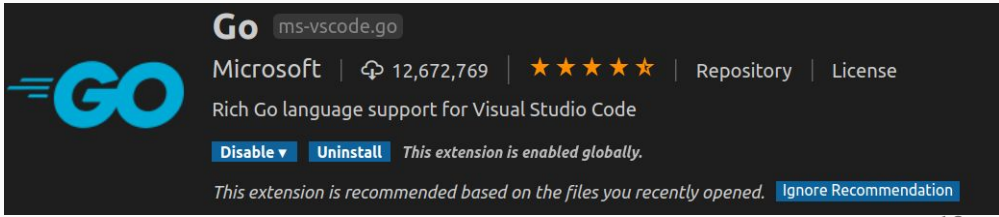
Most popular editors have some kind of support for Go programming language

You can use **Visual Code Editor** by adding Go extensions

https://code.visualstudio.com/

Extensions to add

Go to the extension marketplace on Visual Code editor and install Go extension shown below



Go Code Organization

- Go programmers typically keep all their Go code in a **single workspace**.
- A workspace contains many version control repositories (managed by Git, for example).
- Each repository contains one or more packages.
- Each package consists of one or more Go source files in a single directory.
- The path to a package's directory determines its import path.

Go Code Organization: **Workspace**

Go to your home directory and check if you have a directory named **go**

On Linux or Mac OS it is located

```
$HOME/go
```

On Windows it is found

```
C:\Users\YourName\go
```

You might need to create it, if it is not already there

GOPATH environment variable

- Run `go env GOPATH` command to check the the default workspace value stored in GOPATH env variable
- If you put the go directory somewhere else you need to set the GOPATH env variable

Directories inside the Workspace

Typical go code directory structure looks like the one shown in the right

Create the **`bin/`** and **`src/`** **`subdirectories`** inside your workspace (go directory) if they are not already there

The src subdirectory typically contains multiple version control repositories (such as for Git) that track the development of one or more source packages

```
bin/
    hello                          # command executable
    outyet                         # command executable
src/
    github.com/golang/example/
        .git/                      # Git repository metadata
        hello/
            hello.go               # command source
        outyet/
            main.go                # command source
            main_test.go           # test source
        stringutil/
            reverse.go             # package source
            reverse_test.go        # test source
    golang.org/x/image/
        .git/                      # Git repository metadata
        bmp/
            reader.go              # package source
            writer.go              # package source
```

Setting up workspace

- **Import path** is a string that uniquely identifies a package
- Corresponds to its location inside the workplace / remote repository
- Packages from standard libraries are given short names like “fmt” but for your own library you must choose a base path that is unique
- Using your github /gitlab username create the base path for your project
  - github.com/___username_____
- Create the folder for your program
  - helloworld
-

Hello world

- Go files have file extension of ".go"
- Every go files needs to be included in a package
- **main package main function** is used as the entry point of your application
- To build use the following command

```
go build helloworld.go
```

- To run use `./helloworld`
- Or use `go run helloworld.go`

```go
package main

import ("fmt")

func main() {

    fmt.Println("Hello World")

} // Print | Println ?
```

helloworld.go

Hello world V2.0

- By convention package name is same as the last element of the import path
- Import can also be separated by break or can be written as multiple line imports
- Names are exported if it begins with capital letters, when importing a package you can only refer to the exported names

```
package main
import ("fmt";"math/rand")
func main() {
    fmt.Println("My favorite number is",
rand.Intn(10))
    //fmt.Println(math.pi)
    /* what is the output ? */
}
```

helloworld.go

Hello world V3.0

- `go install`
  `gitlab.com/natget21/helloworld`
  `Or go install inside the package`
- //build your program and installs it in bin
- initialize/add/commit to git
- Go build stringutil
- `go get github.com/golang/example/hello`

Basics of the language (variables)

- **var** statement declares list of variables
- Types are stated at last if variable is initialized type can be omitted
- Inside a function **:=** can be used as short statement instead of var

```
package main
import "fmt"
var i, j int = 1, 2
func main() {
    var c, python, java = true, false, "hi"
    K := 3
     fmt.Println(i, j, c, python, java)
}
```

variables.go

Basics of the language (variables)

```
Bool   , string   , byte

int  int8  int16  int32  int64

uint uint8 uint16 uint32 uint64

rune // alias for int32  represents a Unicode

float32 float64

complex64 complex128
```

Zero values: Uninitialized variables will have

- Bool - false
- Int - 0
- String - “”

Basics of the language (variables)

- Type conversion

  T(V) : Converts value V to type T

- Constants

  Declared using **const**

  const pi = 3.14 *//boolean ,int,string*

```
package main

import ("fmt";"math")

func main() {

    var x, y int = 3, 4

    var f float64 = math.Sqrt(float64(x*x + y*y))

    var z uint = uint(f)

    fmt.Println(x, y, z)

}
```

variables.go

Basics of the language (functions)

```
package main

import "fmt"

func add(x int, y int) int {return x + y}

func add2(x, y int) int {return x + y}

func swap(x, y int) int {return y , x}

func main() {fmt.Println(add(42, 13))}

//types of parameters needs to be defined

// functions can return multiple results
```

```
func split(sum int) (x, y int) {

    x = sum * 4 / 9

    y = sum - x

    return

}

//function can return multiple results

//function can have named return values
```

Basics of the language (flow control)

```
for i := 0; i < 10; i++ {
    sum += i
}
// No ( ) unlike c,javascript
for sum < 1000 {
    sum += sum
}
// init and post statment are optional
// becomes while
for {
}
// infinite loop
```

```
if x < 0 {
    return (x+x)
}
//no brace needed
if v := math.Pow(x, n); v < lim {
    return v
}else{
    return lim
}
//if with short statement
//variables declared here last until the
 end of the if-else
```

Basics of the language (flow control)

```
fmt.Print("Go runs on ")
switch os := runtime.GOOS; os {
case "darwin":
    fmt.Println("OS X.")
case "linux":
    fmt.Println("Linux.")
default:
    fmt.Printf("%s.\n", os)
}
//no break statement
```

```
t := time.Now()
switch {
case t.Hour() < 12:
    fmt.Println("Good morning!")
case t.Hour() < 17:
    fmt.Println("Good afternoon.")
default:
    fmt.Println("Good evening.")
}
// same as switch true no condition
```

Basics of the language (flow control)

- Defer - a defer statements defers (postponds) the execution until the surrounding function returns
- Multiple defer statements in a function will be pushed to a stack and are executed in last in first out order

```
// what is the output ?
```

```
package main

import "fmt"

func main() {
    fmt.Println("counting")
    defer fmt.Print("end")
    for i := 0; i < 10; i++ {
        defer fmt.Println(i)
    }
    defer fmt.Println("done")
}
```

Assignment 1

- Write a program that prints a multiplication table for numbers up to 12.
- Write a program that prints diamond shape using “ * ” with diagonal of 10 “ * “ decreasing by 1 “ * ” per row

Grading : clean , efficient , implementation of the basics

- Write a guessing game where the user has to guess a secret number. After every guess the program tells the user whether their number was too large or too small.

  Number should be 1 - 20 and should be no more than 5 tries , At the end the number of tries should be printed.

  counts only as one try if they input the same number multiple times consecutively.

Introduction to Go (lab 1)

End