trainologic

Logging

trainologic

Logging

- Log is a standard library of Go
- It provides us with methods to write to a designated writer
- By default, writes to stdout (can be changed)
- It is pretty basic. No log level control is provided
  - Uber Zap - https://github.com/uber-go/zap
  - Google logger - https://github.com/google/logger

trainologic

Error Handling

trainologic

Introduction

- Go does not have exceptions
  - The closest thing is panic (Will discuss later)
  - https://golang.org/doc/faq#exceptions
- The idiomatic way to deal with errors is to treat them as values

```
func main() {
        _, err := os.Open("no-file.txt")
        if err != nil {
                fmt.Println("err happened", err)
        }
}
```

trainologic

Dealing With Errors

- There are a few ways to deal with errors
  - fmt.Println(err) - prints the error to Stdout
  - log.Println(err) - prints to Stdout (can be diverted to a file)
  - log.Fatalln(err) - prints to log output and then exits the process with code 1
  - panic(err) - Will discuss later on

trainologic

Custom Error Types

- Errors.New(str) - is the idiomatic way to create new errors
- Allows us to create our own custom errors
  - Variable name should start with Err

```
var ErrSpaceOdyssey = errors.New("My Cool Error Message")
```

trainologic

Customize Errors

- Besides using errors.New() we can satisfy the Error interface and provide even more context
- Let’s write an example

trainologic

Panic & Recover

- For error-handling, Go provides two keywords: *panic* and *recover*.

trainologic

Panic

- When *panic* is called, normal execution stops and the function returns to the caller.
- Of-course, *defer* statements are still executed.
- At the caller site, the function that returned is behaving like a direct invocation of *panic*.
- So, it will continue until the stack of the goroutine rolls all the way back and the program crashes.

- Unless *recover* is invoked!

trainologic

Recover

- recover is a built-in function that regains control of a panicking goroutine.
- Is only usable in *defer* statement.
- Returns the value provided to the *panic* function or *nil* if not panicking.

trainologic

Flags

trainologic

Flags

- Go provides a useful standard library that allows to add flags to program execution
- Usage is pretty straight forward
- Let's see an example..

```
package main

import "flag"
import "fmt"

func main() {

    wordPtr := flag.String("word", "foo", "a string")

    numbPtr := flag.Int("numb", 42, "an int")
    boolPtr := flag.Bool("fork", false, "a bool")
```

trainologic

Flags

```
func main() {

    wordPtr := flag.String("word", "foo", "a string")

    numbPtr := flag.Int("numb", 42, "an int")
    boolPtr := flag.Bool("fork", false, "a bool")

    flag.Parse()

    fmt.Println("word:", *wordPtr)
    fmt.Println("numb:", *numbPtr)
    fmt.Println("fork:", *boolPtr)
}
```

trainologic

Exercise Time

trainologic

Exercise

- Build a simple cli tool named search
- The application searches for all files with given extention that contains a specified string under the current directory
- Usage should be: search -ext [ext] -text [text]