Introduction: Hello World, values, and variables

Go code layouts

A Go project is also called a module. A module is just a collection of packages.

A package is just a group of related .go files. You would declare the .go files that belong in the same package with the line

```
package <package name>
```

For example: If you use the `main` package it is used to make the package an executable program (you get a binary) because it contains your main function. The `main` package tells the Go compiler that the package will be compiled as an executable program rather than a library which will not produce an executable.

Otherwise, the package name can be whatever you want. However, keep the package name that you are declaring the same as the directory that it is under. For example:

```
src
helper
    - helper.go
    main.go
```

If you have a directory like such keep the package name that you use in `helper.go` as `helper` because if do `package lol` which doesn't match the directory name. You would be importing the helper package in `main.go` as

```
import (
"module/path/helper
)
```

But when you want to call the function from the `lol` package it would be

```
lol.helperFunc()
```

So keeping the directory name and the package name the same would make it easier for yourself and for others to maintain.

Hello World

```
package main

import "fmt"

func main() {

    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)
}
```

As you can see the `fmt` module that is imported is the built-in module in Golang for printing things out to the consoles.

`Println` is just one of the functions inside `fmt` module to print things, there are many others.

> Notice that the function is capitalized. This is not arbitary but rather telling Go that the Println function from the `fmt` module is exported for others to use. If it is lower-case then it cannot be used by others.

Variables

Couple of ways of declaring a variable:

1. This is the most verbose way of declaring and assigning it, you have the type of the variable on the right and then assignment to the variable of the expected type to the right of the assignment operator

```
var x int = 10
```

2. A simpler variable declaration and assignment of the previous method, you can skip out on the type if the type on the right is what you want of the assignment. This is possible because the compiler can infer the type in compile time

```
var x = 10
```

3. If you just want to declare the variable without assigning it here it is. The variable will have it's zero value assigned.

```
var x int
```

4. You can also declare multiple var in a block. Also provide them with initialization as well

```
var (
	x int
	y int
	z string = "default"
)
```

5. Go also have a short hand declaration that you can only use within a function. This method cannot be used outside of the function, but `var` can be. You can just skip out the `var` keyword and add in a colon. The compiler will use type inference to infer the type on the right.

```
x := 10
```

6. There is also constants in Go but they are pretty limited. You can only assign them the values that compiler can figure out at compile time. Type with constant is allowed as well.

```
const y = "hello world"
```

var vs :=

There are limitation on `:=`. Such as you cannot use it outside of functions to declare a global variable like you can with `var`.

You cannot have a zero value if you are using `:=`. You must assign something to it in order to make it work.

Zero value

Type	Zero value
Boolean	false
Numeric types	0
Float	0
String	"" (Empty string)
Structs	Each of struct's key is set to their respective zero-value

Revision #3
Created 2023-05-26 02:35:25 UTC by Tamarine
Updated 2023-05-29 03:26:05 UTC by Tamarine