Names: Charles Shook, Mark Weiss, Christopher Bare, Trevor Bormann

Language: Golang

Coding environment: <1.7, Windows 10, Visual Studio Code, https://go.dev/>

Distinct pieces of functionality:

1. <Help Command> : <commands.go>
2. <Chicken Command> : <commands.go>
3. <Date Command> : <commands.go>
4. <Add Command> : <commands.go>
5. <Sub Command> : <commands.go>
6. <Divide Command> : <commands.go>
7. <Play Command> : <commands.go>
8. <List Role Command> : <commands.go>
9. <Change Role Color Command> : <commands.go>
10. <Stats Command>: <commands.go>

Section 1: Comparison to C++, Java, Python, and/or Scala

1. **What is the language's philosophy?**

   Go, which is also referred to as Golang, was born out of the frustration of working with other languages like C++ or Java. Setting up the environment to work with these languages was extremely difficult do. When sharing projects this caused a lot of issues as development environments were difficult to replicate from one system to another. Go's goal was to combine that best part of all languages like the ease of use of interpreted languages, the convenience of garbage collection and memory safety, and the efficiency and reliability that comes with statically typed languages.

   Go's designers envisioned making this language modern by adding great networking and multicore computing. They also envisioned Go being fast to compile. The goal was to have a large project build in a few seconds using only a single computer. To achieve this, Go needed to improve the type system, concurrency, garbage collection, rigid dependency specification, and much more.

2. **Compare and contrast your language in terms of the location it is used.**

Go was built extremely lightweight and includes a lot of networking features which made it popular for cloud and server-side applications. Due to Go's speed and it being lightweight it is being used more CLI applications that need to be as fast as possible. Go is also starting to be used in DevOps because of its speed.

We see Scala used a lot for data procession which it is good for due to its functionality, but you would choose Go in a situation that required speed. You see the same with Java where it is used for GUI applications but once again you would choose Go over Java if you wanted the highest amount of speed possible.

3. **Compare and contrast your language in terms of where it excels and where it fails**
   - **Excels**

Golang excels in areas that commonly require concurrency and network integration. Concurrency in Go is cheaper than creating a tread in other languages. Due to this you can have thousands of go routines in a single project. For all these to comminate go added a channel system to go routines to communicate.

Go's code compiles extremely fast. It compiles so fast that it almost looks like it's an interpreted language like Python. However, it is just a compiled language. Go's creators wanted to make the compilation process extremely fast, and you can compile a Go project with thousands of files in seconds.

Go's module and package system is quite nice and robust. In Go you can have a package in a public or private GitHub repo that you can then import into your project. GitHub is used by a majority of develops who host their code in GitHub, so this integration is quite nice to use.

Lastly, it is widely open source, allowing creators to make tools, IDE's and plugins that can implement Go for more programmers to use.

   - **Fails**

Golang fails in a few different places, mainly with functions. Go has no compatibilities with function overloading nor with default parameters, however the default parameters can be dealt with and managed in the same way as Java. Another area that could be improved is in generics. If you wish to create your own map function for an array of integers and want to apply the same function to an array of strings, you will need a second function.

Go also doesn't really have OOP where Python, Java and Scala did. This can make working with Go difficult initially because we have been thought with these other languages to think about OOP for everything. So, in that aspect Go fails to reach the level of these other languages we have looked at so far.

4. Compare and contrast your language in terms of Portability, Simplicity, Orthogonality, AND Reliability.

- **Portability**

Golang is a compiled language, and the created binaries are platform specific, so once compiled Golang is not a portable language. However, the Golang compiler is capable of cross-platform compilation for different platforms and architectures using the same source code and machine which allows for easy portability to most operating systems. The process of building for different platforms is quite simple as only a few build flags need to be set for it to compile.

- **Simplicity**

Simplicity in Go comes mainly from the dependencies or imports, as well as the straightforward typing. Meaning there are very few dependencies needed for what you will end up needing to do. This is similar to other languages, but still worth pointing out. The straightforward typing means it gets rid of a lot of unnecessary words. For example, if you wish to export a function from a file, you simply capitalize the function. Initializing a variable just requires a colon before the equals and will initialize the variable to the type of the value specified using type inference. Lastly importing dependencies is as simple as saying import("dependency")

- **Orthogonality**

Orthogonality in Go is solid, it follows some of the same issues as C++. This means that in general a '+' is used to add one thing to another, and an '=' is used to set them equal. However, Go syntax is heavily based on C and has some of the same orthogonality pitfalls. For instance, '*' is used both for pointer declaration and multiplication and '&' is used for both memory access and binary AND, which are completely unrelated operations.

- **Reliability**

According to an article from Ben Congdon, through all his coding, and many years of experience using Go, there has not been a single update that has broken any of his code. Going along with that, most dependency libraries that are accessible through Go are stable. Meaning if they haven't been updated in years, it's because they work, not because someone has stopped updating them.

Section 2: Syntax, OOP

1. **A) Write an example of one type of assignment expression in the language.**
   B) Then write the EBNF for a generic version of this assignment expression with all tokens defined.

a) variable := assigned value. Ex: x := 25

Note: A Go program will only compile if all variables are used. Any unused variable causes the program to not compile

b)

<lower> ⇐ a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<upper> ⇐ A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<letter> ⇐ <upper>|<lower>

<digit> ⇐ 0|1|2|3|4|5|6|7|8|9

<symbol> ⇐ `|~|!|@|#|$|%|^|&|*|(|)|-|=|+|_|\|||{|}|[|]|:|;|'|?|/|>|.|<|,

<sign> ⇐ -|+| Є

<identifier> ⇐ (<letter>|_) | (<identifier> (<letter> | <number> | _ | Є))

<string> ⇐ (“{<symbol>|<digit>|<letter>}”) | (‘{<symbol>|<digit>|<letter>}’)

<number> ⇐ {<digit>}[. {<digit>}]

<Assignment> ⇐ <identifier> := (<string> | <number> | <identifier>)

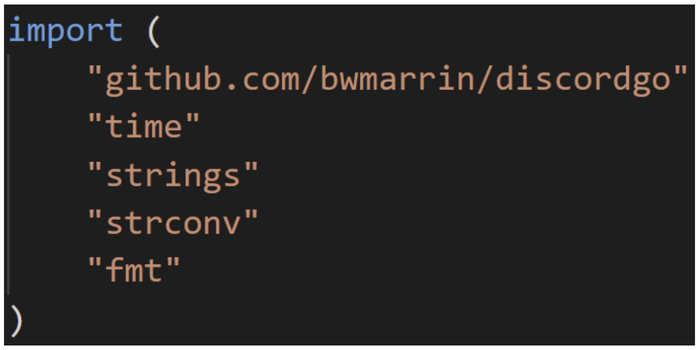
2. **A) How does the language support extension etc. (single inheritance, interfaces, root object, class OOP, prototype OOP, other OOP, file importing, file extension, plugins, piping, module linking, etc.)?**

Go doesn’t really support OOP like we have seen in Scala, Python and Java so far, but you can sort of fake it. Go supports Structs which can have functions linked to them, like how you could use structs in C or C++. However, there is no inheritance in Go. Since Go was built to run on servers, they were looking for speed, so they left out many of the more computationally inefficient aspects of object oriented programming that would have slowed the language down.

Go also supports interfaces. When defining an interface, you also define a set of functions with inputs and outputs defined. Any struct that implements these functions will be considered part of the interface you defined. So, you could define an animal interface with eat and sleep functions and then have a dog or cat struct that implements the eat and sleep functions. It will then be considered an animal. So, any function that takes in the animal interface will also take in the cat and dog struct since they have the required functions.

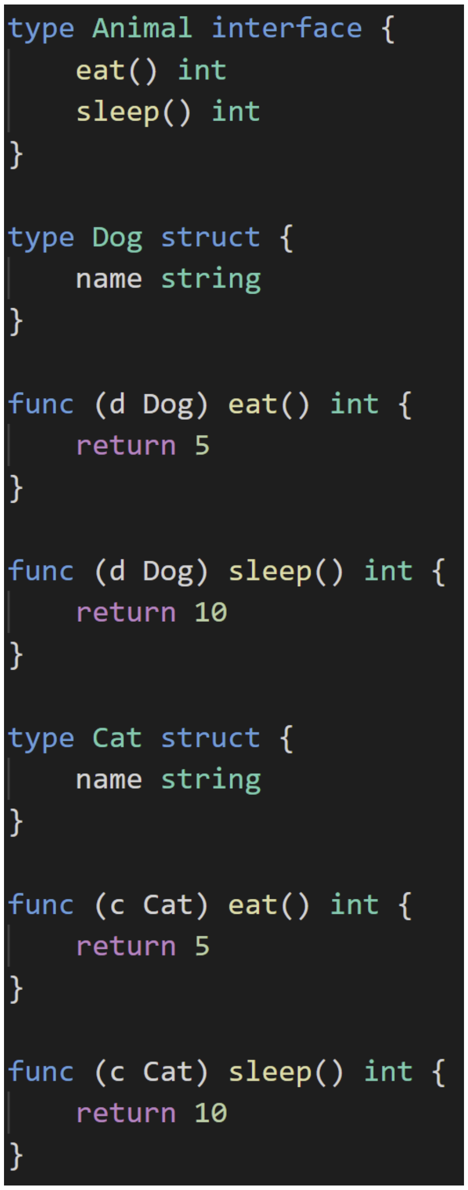
Go uses packages and modules to allow for ease of use when pulling in code from other sources. They have a nice system for hosting code in GitHub, both in public repositories and private repositories. You are then able to import code from these repositories into your project. Go modules are what handles all the dependencies in your project, and it is what allows others to import your package. You then create different packages under that module and users can use the packages of their choosing.

B) Give an example.



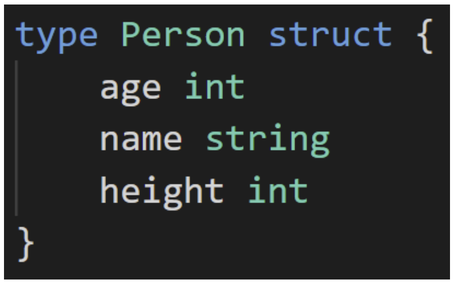
```go
import (
	"github.com/bwmarrin/discordgo"
	"time"
	"strings"
	"strconv"
	"fmt"
)
```

Figure 1.1 Importing packages in Golang



```go
type Animal interface {
	eat() int
	sleep() int
}

type Dog struct {
	name string
}

func (d Dog) eat() int {
	return 5
}

func (d Dog) sleep() int {
	return 10
}

type Cat struct {
	name string
}

func (c Cat) eat() int {
	return 5
}

func (c Cat) sleep() int {
	return 10
}
```

Figure 1.2 Interface example in Go



```
type Person struct {
    age int
    name string
    height int
}
```

Figure 1.3 Struct example in Go

3. **A) How does the language handle module/namespace/packages/etc.**

   Go handles these items all the same, you add them to a list of imports at the top of the file in the style of "import (x, y, z)" where x, y, and z can be anything ranging from other files, pre-installed packages, or even GitHub links. An example of this can be seen in Figure 1.1, where the discordgo library is imported from GitHub as well as some other packages.

   B) What is the scope operator(s)? Alternatively, how to pick which variable if two code courses contain the same name?

   Since Go is a block scoped language, there is no way to decide between inner and outer scope variable with the same name. If there are two variables with the same name, it'll choose the closest variable in terms of scope.

   C) Does the language allow function overloading (name repetition), function redefinition, and/or function overriding?

   Go does not support function overloading or name redeclaration. They did this because it was seen in other languages to be useful some of the time, but ultimately caused confusion as people could redefine the operators to do what they want and there was no consistency, breaking orthogonality. Go instead would rather you write your own function that you would then call yourself rather than user operator overloading.

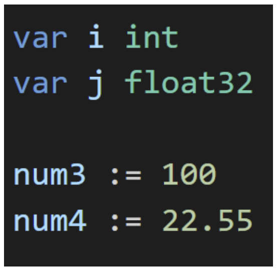
   D) Give example <u>syntax</u> if it does.

   Go does not support function or operator overloading, name repetition, function redefinition, or function overriding.

Section 3: Binding, Type system, and data type range

1. **Is the language static or dynamically typed? Give example <u>syntax in code</u>.**

   Golang is a statically typed language.

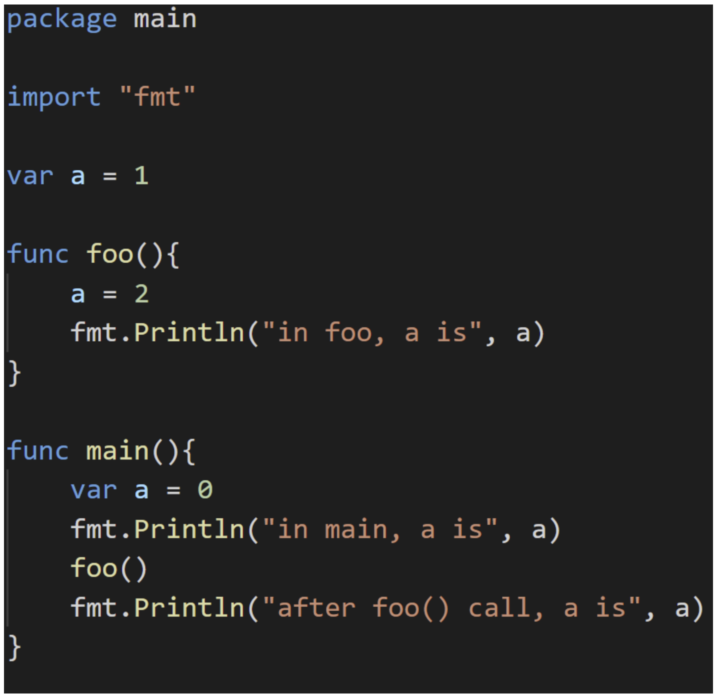


```go
var i int
var j float32

num3 := 100
num4 := 22.55
```

Figure 1.4 Static typing example

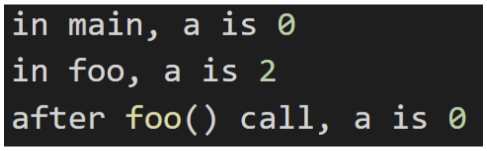
2. **Is the language static or dynamically scoped? Give an example <u>in code</u>.**

   Go is lexically scoped with blocks.



```go
package main

import "fmt"

var a = 1

func foo(){
    a = 2
    fmt.Println("in foo, a is", a)
}

func main(){
    var a = 0
    fmt.Println("in main, a is", a)
    foo()
    fmt.Println("after foo() call, a is", a)
}
```

Figure 1.5 Scoping example

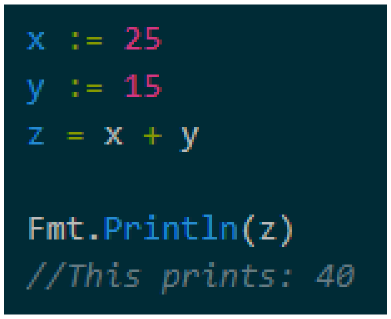


```
in main, a is 0
in foo, a is 2
after foo() call, a is 0
```

Figure 1.6 Output of scoping example

3. **How is the language read (in-fix, pre-fix, a mix, etc)? Give an example <u>in code or in a diagram</u>. (Max of 3 examples).**

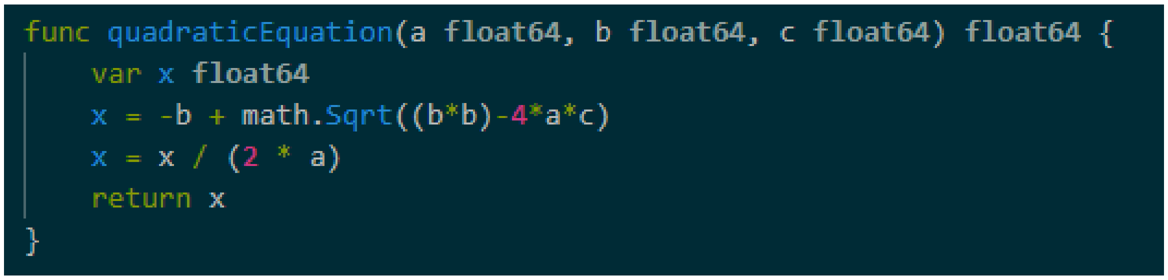
   Golang is read as in-fix to increase readability, and programming efficiency.



```
x := 25
y := 15
z = x + y

Fmt.Println(z)
//This prints: 40
```

Figure 1.7 in-fix reading example

Quadratic function



```
func quadraticEquation(a float64, b float64, c float64) float64 {
    var x float64
    x = -b + math.Sqrt((b*b)-4*a*c)
    x = x / (2 * a)
    return x
}
```

Figure 1.8 in-fix typing example of the quadratic Function

4. **What are the built-in data types and their ranges? (List 4-10, or send me a note if you believe that are less than 4)**
   - Uint8 (0 to 255)
   - Uint16 (0 to 65535)
   - Uint32 (0 to 4294967295)
   - Uint64 (0 to 18446744073709551615)
   - Int8 (-128 to 127)

- Int16 (-32768 to 32767)
- Int32 (-2147483648 to 2147483647)
- Int64 (-9223372036854775808 to 9223372036854775807)
- Float32 (IEEE-754 32-bit)
- Float64 (IEEE-754 64-bit)

References

Frequently asked questions (FAQ). Go. (n.d.). Retrieved December 10, 2021, from https://go.dev/doc/faq.

9, E. P. D., Pulsifer, E., 9, L. K. D., & Klint, L. (2021, September 28). *What is go? an intro to google's go programming language (aka Golang)*. A Cloud Guru. Retrieved December 10, 2021, from https://acloudguru.com/blog/engineering/what-is-go-an-intro-to-googles-go-programming-language-aka-golang.

Zhang, K. (n.d.). *The Go Programming Language Report*. Dynamic scope or static scope | The Go Programming Language Report. Retrieved December 10, 2021, from https://kuree.gitbooks.io/the-go-programming-language-report/content/24/text.html.

Go - data types. (n.d.). Retrieved December 10, 2021, from https://www.tutorialspoint.com/go/go_data_types.htm.

Ben Congdon. (2020, January 11). *The value in go's simplicity*. Ben Congdon. Retrieved December 10, 2021, from https://benjamincongdon.me/blog/2019/11/11/The-Value-in-Gos-Simplicity/.