Go (GoLang)

By: Dominic Theis and George Goodman

Background

- Designed by Robert Griesemer, Rob Pike, and Ken Thompson in 2009
- Done as an experiment to improve common languages such as C, Java, and C++
  - Improve negative aspects while keeping the good aspects
- Used as a systems programming language for large distributed systems and network servers
- Implemented to replace C++ and Java on Google's software stacks
- Built for highly scalable networks, and to reduce the lines of C++ code
- Other users:
  - Youtube
  - Santa Tracker
  - Adobe

https://sdtimes.com/webdev/more-developers-using-go-programming-language/

General Info

- Compiled, Statically typed, much like Algol and C
- Garbage Collection
- Scalable to large systems
- Cleaner and simpler language meant to reduce redundancy and maintain readability
- Supports network and multiprocessing
- Uses command line mostly

Types

- Numeric: byte, int64, float32
- Booleans
- Character strings (String)
  - Strings are immutables
- Built in operators
- Structs
- Dynamic Arrays = “Slices”

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

```
q := []int{2, 3, 5, 7, 11, 13}
fmt.Println(q)
```

```
r := []bool{true, false, true, true, false, true}
fmt.Println(r)
```

Syntax

- Aimed to keep code concise and readable
- Uses combined declaration/initialization
  - Go will infer the variable type
- Constants
- Semicolons terminate statements
- Go's range replaces C's loops

```
const georgeIsCool := true
```

```
var George = "cool"
```

```
George := "cool"
```

```
# command-line-arguments
./Functions.go:12:9: no new variables on left side of :=
```

```
for i, num := range nums {
    if num == 3 {
        fmt.Println("index:", i)
    }
}
```

```
George, Dom := "cool", true
```

Control Structures

- For Loops
  - The most basic for loop initializes a variable to increment, then runs the loop
  - Uses the shorthand like java with the an initializer, condition and the after in one statement
- If Statements
  - Parentheses are not necessary around conditions
  - Braces are required for if and else statements
  - Can have a single if statement
  - Any variable declared in an if can be used in any branch

```
odd
even
divisible by 4
odd
12 has multiple digits
```

```go
package main
import "fmt"
func main() {
  for j :=7;j<=9;j++ {
    if j%2 == 0 {
      fmt.Println("even")
    }
    else {
      fmt.Println("odd")
    }
    if j%4 == 0 {
      fmt.Println("divisible by 4")
    }
  }
  if num:=12; num <0 {
    fmt.Println(num, "is negative")
  }else if num < 10 {
    fmt.Println(num, "has 1 digit")
  } else {
    fmt.Println(num,"has multiple digits")
  }

}
```

Maps

- Use make to create a map
- Set key and value pairs using name[key] = val syntax
- fmt.Println will show all key/value pairs
- Get a value for a key with name[key]
- Built in len function gives you the number of pairs
- Built in delete function removes key/value pairs
- Declaration and initialization shortcut available in same line syntax

```
map: map[Dominic:21 George:22]
name1: 21
len: 2
map map[Dominic:21]
map: map[foo:1 bar:2]
```

```go
package main
import "fmt"
func main() {
  m:=make(map[string]int)
  m["Dominic"] = 21
  m["George"] = 22
  fmt.Println("map:", m)
  name1 := m["Dominic"]
  fmt.Println("name1:", name1)
  fmt.Println("len:",len(m))
  delete(m,"George")
  fmt.Println("map",m)

  n:=map[string]int{"foo":1, "bar":2}
  fmt.Println("map:",n)
}
```

Functions

- Functions
  - Indicated by the func keyword
  - Take zero or more parameters and can return zero or more values
  - Error handling as return

```
1+2= 3
1+2+3= 6
errorTest Failed can't work with 42
```

```go
package main
import "fmt"
import "errors"
func plus(a int, b int) int {
  return a + b
}
func plusPlus(a,b,c int) int {
  return a + b + c
}
func errorTest(arg int) (int,error) {
  if arg == 42 {
    return -1,errors.New("can't work with 42")
  }
  return arg + 3, nil
}
func main() {
  res:=plus(1,2)
  fmt.Println("1+2=",res)
  res = plusPlus(1,2,3)
  fmt.Println("1+2+3=",res)
  if r, e  := errorTest(42); e !=nil {
    fmt.Println("errorTest Failed",e)
    } else{
      fmt.Println("errorTest passed", r)
    }
}
```

Interface System

- Interfaces
  - Designed after protocols from smalltalk
  - Provide runtime polymorphism
  - Provide limited form of structural typing
  - Any type that implements all methods of an interface conforms to that interface
  - Allows converting interface values to other types with a runtime type check

```
package main

import "fmt"
import "math"

type geometry interface {
    area() float64
    perim() float64
}

type rect struct {
    width, height float64
}
type circle struct {
    radius float64
}

func (r rect) area() float64 {
    return r.width * r.height
}
func (r rect) perim() float64 {
    return 2*r.width + 2*r.height
}

func (c circle) area() float64 {
    return math.Pi * c.radius * c.radius
}
func (c circle) perim() float64 {
    return 2 * math.Pi * c.radius
}

func measure(g geometry) {
    fmt.Println(g)
    fmt.Println(g.area())
    fmt.Println(g.perim())
}
```

GoRoutines

- GoRoutines take up dead time
  - especially with networking on Google servers
- Concurrency
- Use go to invoke a function in a goroutine
- GoRoutines execute concurrently with calling
- Go function calls are run asynchronously
  - Scanln requires a key press before the program exits
- GoRoutines printed last

```
$ go run goroutines.go
direct : 0
direct : 1
direct : 2
goroutine : 0
going
goroutine : 1
goroutine : 2
<enter>
done
```

```
paclage main
import "fmt"
func f(from string) {
  for i:=0; i<3;i++ {
    fmt.Println(from, ";", i)
  }
}func main() {
  f("direct")
  go f("goroutine")
  go func(msg string) {
    fmt.Println(msg)
  } ("going")
  fmt.Scanln()
  fmt.Println("done")
}
```

Omissions & Criticisms

- Omissions
  - Inheritance
  - Assertions
  - Pointer Arithmetic
  - Implicit type conversions
- Criticisms
  - Lack of compile time generics lead to code duplication
  - Lack of exceptions make error handling more difficult
  - Pauses and overhead of garbage collection limit use in systems programming

Sources

- https://tour.golang.org/list
- https://en.wikipedia.org/wiki/Go_(programming_language)
- https://gobyexample.com/range
- https://www.youtube.com/watch?v=Zg7GK759ZzA
- https://blog.golang.org/go15gc