Intro to Go

Women Who Go NYC

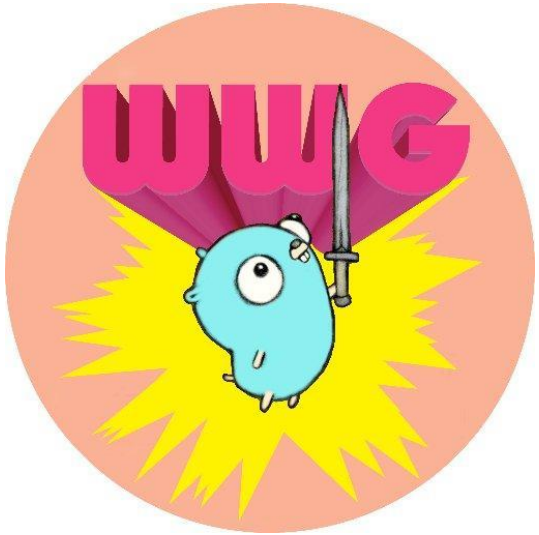
February 2020

Women Who Go

Women Who Go is an inclusive space for women, non-binary and transgender people with interest in the Go programming language, or programming in general. Women Who Go works to build a more diverse and inclusive golang community.

https://www.meetup.com/NYC-Women-Who-Go

@womenwhogo_nyc



Class Resources

The follow resources will be used for this course:

- The slides and sample exercise solutions are on Github
- Go installation guide: https://golang.org/doc/install
- All exercises can be completed in the Go Playground
- The course is based on the training by GoBridge organizer Johnny Boursiquot
- Overview and examples are based on Go by Example

Questions? Find me on Twitter @yigenana

Class Goals

Review language-agnostic concepts like variables, conditionals, looping, etc and demonstrates their use with the Go language syntax. Work through exercises to apply new concepts using Go.

- Working with variables and constants
- Working with conditionals, if/else statements
- Working with lists and looping
- Using functions for reusing functionality
- Using complex types to represent concepts out in the real world

Agenda: Day 1

6:30 - 6:45: Introduction to workshop and class

6:45 - 7:15: Values, variables, constants, for loops, if/else, switch

7:15 - 7:45: Arrays and Slices

7:45 - 8:00: Exercise

Agenda: Day 2

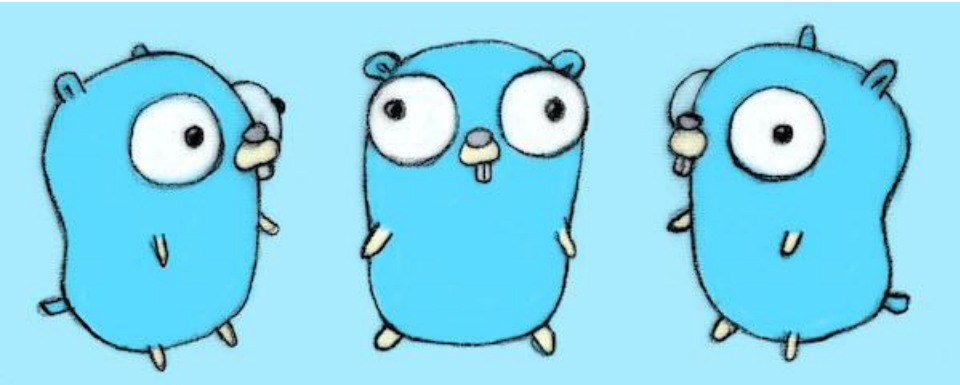
6:30 - 7:00: Map, Range, Function

7:00 - 7:40: Exercise

7:40 - 7:50: Future Learning and Resources

7:50 - 8:00: GoRoutines

Go Programming Language



The Go programming language is an open source project to make programmers more productive.

Key Features

- Static
- Compiles
- Garbage Collected
- Native Concurrency Support

Values, variables, constants, for loops, if/else, switch

Lesson 1 - 7

Let's Get Started!

Lessons:

https://jboursiquot.gitbooks.io/learn-to-code-with-go-workshop/lesson-1.html

Go by Example:

https://gobyexample.com/hello-world

Go Playground:

https://play.golang.org/

Lesson 1: Hello World!

Challenge

fmt.Println automatically puts a newline at the end of your string. Rewrite your program to say hello the person sitting next to you.

Advanced Challenge

Install Go on your local computer. Go can be downloaded at https://golang.org/dl/.

Install our editor of choice: Visual Studio Code. (Or use your preferred editor)

Copy your simple program from the Challenge above and run it from your editor.

Lesson 2: Values

Go has various value types. The basic types are:

- Strings: “This is a string”
- Integers: 42
- Floats: 3.14
- Booleans: true

Why might data types be useful in a programming language?

What kinds of things could you not represent with these four types?

Lesson 2: Exercise

Challenge

Get your program to print out the following sentence:

Hi, I'm NAME and I am AGE years old. My favorite color is COLOR and I have $X.XX in my pocket!

Each time there's something in CAPITALS, use a comma and use the right Go type.

For example: fmt.Println("My favorite color is ", "blue")

Advanced Challenge

Do the challenge again, this time, using fmt.Printf and the appropriate format verbs.

Lesson 3: Variables

Variables are for storing data that can be used later on within a program.

In Go, variables are explicitly declared and used by the compiler to e.g. check type-correctness of function calls.

Variables in Go

- Keyword “var” declares 1 or more variables.
- You can declare multiple variables at once.
- Go will infer the type of initialized variables.
- Variables declared without a corresponding initialization are zero-valued.
- The := syntax is shorthand for declaring and initializing a variable.

Lesson 3: Exercise

Challenge

Update the challenge from Lesson 2 to use variables for the following values name, age, color, money

Advanced Challenge

In Go, it is common to assign the results of a function to one or more variables. For this challenge update the code from above to use fmt.Sprintf which will return a string version of the formatted output, and store the return value to a new variable named “msg”.

Lesson 4: Constants

Go supports constants of character, string, boolean, and numeric values.

The keyword “const” declares a constant value.

The value of a constant cannot change during the program’s execution, while the value of a variable can.

Lesson 4: Exercise

Challenge

Go has many constants already defined, for example `math.Pi`.

Print out the perimeter distance around a circle that has a radius of 5 feet.

The formula for perimeter distance is `2 * math.Pi * radius`.

Advanced Challenge

Update the code above to also output the area of the same circle. The formula for area is `math.Pi * radius^2`

Lesson 5: Loops

The keyword “for” is Go’s only looping construct. Loops provide a initial state that starts the loop, a condition that stops the loop, and an action that is performed for each loop iteration.

Loops can also be structure to have a single stopping condition and update the variable state during the loop iteration.

```
for j := 7; j <= 9; j++ {
        fmt.Println(j)
    }


var i = 1
    for i <= 3 {
        fmt.Println(i)
        i = i + 1
    }
```

Lesson 5: Exercise

Challenge

Write a for loop that only executes once. Then write a for loop with no condition statement that exits when the count is equal to 100.

Advanced Challenge

Use the for loop to write a program that outputs a right isosceles triangle of height and width n, so n = 3

```
*

**

***
```

Lesson 6: If/Else Statement

Go provides the “if” and “else” keyword to create conditional logic in your program.

You can also have an If statement without an Else

```
  if 7%2 == 0 {
    fmt.Println("7 is even")
  } else {
    fmt.Println("7 is odd")
  }

  if 8%4 == 0 {
    fmt.Println("8 is divisible
by 4")
  }
```

Lesson 6: Exercise

Challenge

Combine a For loop with an If/Else to count to 10 and print out if a number is even or odd.

Advanced Challenge

Combine a for loop with if, else if and else to count to 10 and print whether the number is less than 5, equal to 5, or greater than 5.

Lesson 7: Switch Statement

Switch statements express conditionals across many potential scenarios. It provides an easy way to check a value against many options.

```
t := time.Now()

switch {
case t.Hour() < 12:
 fmt.Println("It's before noon")
default:
 fmt.Println("It's after noon")
 }
```

Lesson 7: Exercise

Challenge

Combine a for loop with switch to count to 10 and print whether the number is less than 5, equal to 5, or greater than 5.

1 is less than 5

2 is less than 5

3 is less than 5

4 is less than 5

5 is equal to 5

6 is greater than 5

7 is greater than 5

8 is greater than 5

9 is greater than 5

10 is greater than 5

Arrays and Slices

Lesson 8 - 9

Lesson 8: Arrays

In Go, an array is a numbered sequence of elements of a specific length.

For example the collection of integers 5, 8, 9, 79, 76 form an array. Mixing values of different types, for example an array that contains both strings and integers is not allowed in Go.

The length of an array is part of its type, thus you cannot change the length of an array.

```
var a [5]int

b := [5]int{1, 2, 3, 4, 5}
```

Lesson 9: Slices

Slices are a wrapper around arrays that provide greater flexibility. Unlike arrays, slices are typed only by the elements they contain (not the number of elements).

Slice support resizing, copying, and slicing from existing arrays and slices.

```
s := make([]string, 3)
    s[0] = "a"
    s[1] = "b"
    s[2] = "c"

s = append(s, "d")

newSlice := s[1:3]
```

Arrays vs Slices

Array

- Length is part of it's type
- Cannot be resized
- Do not need to be initialized explicitly; the zero value of an array is a ready-to-use array whose elements are themselves zeroed
- Arrays are values (as opposed to pointers)

Slices

- Types described by the element they contain
- Have helpful methods to enable resize like operations, like "append" and "slice"
- Need to be explicitly initialized with the "make" operation
- Slices are pointers to underlying arrays and the length and capacity of the slice

Lesson 9

Challenge

Write a program that does the following:

- Creates an empty slice of integers and prints it out -> []
- Loops over the numbers 1 through 10 and appends them to the slice
- Prints out the slice -> [1 2 3 4 5 6 7 8 9 10]
- Prints out the first 5 numbers in the slice -> [1 2 3 4 5]

Day 1: Wrap Up

We've covered a lot!

- Introduction to the Go programming languages
- Values and basic types (string, int, bool, float)
- Variables and constants
- Conditional logic (for loops, if/else, switch statements)
- Complex values: Arrays and Slices

Coming up next:

- Complex values: maps
- Complex logic: range
- Program structure: functions and packages

Intro to Go
Day 2

Agenda: Day 2

6:30 - 7:00: Map, Range, Function

7:00 - 7:30: Project

7:30 - 7:50: Go Routines

7:50 - 8:00: Future Learning and Resources

Map, Range, Function

Lesson 10 - 12

Lesson 10

Maps are Go's built-in associative data type (sometimes called hashes or dicts in other languages).

Maps contain a key and a value. The key and value can be different types, but keys are all the same type and values are all the same type.

The "make" keyword can be used to create an empty map.

```
m := make(map[string]int)
    m["k1"] = 7
    m["k2"] = 13


_, value := m["k2"]
```

Lesson 10

Challenge

Write a program that does the following:

- Creates a map from integers to booleans
- Loops through the numbers 1 through 10 and sets the map where the key is the number and the value is whether the number is even (HINT: use an If/Else or Switch!)
- Prints out the map -> map[1:false 2:true 3:false 4:true 5:false 6:true 7:false 8:true 9:false 10:true]
- Prints out whether 4 is even using variables -> The number 4 is even: true

Advanced Challenge

Write a program that sorts a set of numbers 0 - 30 into two groups: "Numbers divisible by 9", "Numbers not divisible by 9". (Hint: use constants as map keys.) The output for the program should something like the output below:

Lesson 11

The keyword “range” iterates over elements in a variety of data structures and is commonly used to iterate over slices and maps.

When using range, you must assign variables for the item pairs being looped over. In a slice, this is the index and value, in a map, it’s the key and value.

```
nums := []int{2, 3, 4}
sum := 0
for _, num := range nums {
  sum += num
 }

kvs := map[string]string{"a":
"apple", "b": "banana"}

for k, v := range kvs {
  fmt.Printf("%s -> %s\n", k, v)
}
```

Lesson 11

Challenge

Modify your program from lesson 10 to use a range to iterate the map to output the following:

1 is odd 2 is even 3 is odd 4 is even 5 is odd 6 is even 7 is odd 8 is even 9 is odd 10 is even

HINT: use a for loop with range to get the key and value of each entry in the map, then use an If/Else on the value.

Advanced Challenge

When iterating over maps the order of keys is not guaranteed, so running the same program multiple times will result in different outputs, as you may have noticed from the challenged above. To print a map in order the keys must be sorted separately and then used to access the fields of the key.

Update the program above so that the map is always printed in order starting at 1, all they way to 10.

HINT: use the sort.Ints function in the sort package

Lesson 12

All of our examples have been written inside of a “main” function. This main function is required for all Go programs, it is the entrypoint for a Go program and tells Go how to run the program.

A function is a subset of code that generally returns a value. Functions are helpful for organizing programs and making routines reusable in other parts of the program.

```
package main
import "fmt"

func plus(a int, b int) int {
    return a + b
}

func main() {
    res := plus(1, 2)
    fmt.Println("1+2 =", res)
 }
```

Lesson 12

Go programs are organized into packages. Like functions, packages help organize code and enable sharing common functionality. The Go language comes with many built in packages for common programming operations. For example, we've used the "fmt" package to format our code.

Packages are declared at the top of a file. Other packages can be accessed by using the "import" keyword and the package name.

```
package main

import "fmt"

func main() {
  greeting := "Hello gophers!"
  fmt.Println(greeting)
}
```

Lesson 12

Challenge

Write a program that does the following:

- Loop through the numbers 1 through 10
- Call a function that prints out if the number is even or odd (like in Lesson 11).
- Refactor your program to add a new function that takes an integer and returns a boolean signifying if the integer is even, use that function inside your printing function.

Advanced Challenge

Refactor your code so that you end up with a third function. Extra bonus points if you have a function in a function in a function!

Create a Quiz Game

Project

Project

We covered a lot of ground with things like variables, conditionals, and looping. In order to help strengthen those concepts we will write a quiz program that displays (prompts) a question to the user and waits for the user to enter an input (response) to the question being asked. Once all questions have been answered the program should display the total number of questions the user got correct, over the total number of questions asked (i.e You got 5/6 correct).

Hint: You will need a type that can hold both the questions and answers as pairs, and the fmt.Scanln function from the fmt package for reading input from the screen.

- What is 5+5? **10**
- What is 1+1? **2**
- What is 8+3? **12**
- What is 1+2? **3**
- What is 8+6? **14**

You got 4/5 correct!

Next Lessons & Future Learning

Pointers

- Allow you to pass references to values and records within your program.

Structs

- Go's structs are typed collections of fields. They're useful for grouping data together to form records.

Methods

- Go does not have classes. However, you can define methods on types.
- A method is a function with a special receiver argument.

GoRoutines

A goroutine is a lightweight thread of execution used for concurrent programming. Through concurrency, programs run as independent processes working together in a specific composition. Concurrency can help your programs run more efficiently and handle increasing throughput.

Suppose we have a function call f(s). Here's how we'd call that in the usual way, running it synchronously. To invoke this function in a goroutine, use the keyword "go", e.g. go f(s). This new goroutine will execute concurrently with the calling one.

Future Learning & Exercises

Go Basics and Beyond

Concurrency Patterns:
https://blog.golang.org/advanced-go-concurrency-patterns

Channels:
https://www.youtube.com/watch?v=KBZlN0izeiY

Practice Exercises:
https://gophercises.com/