# GOLANG
# 2-Day Training

## SATURDAY

## Introduction

- Front-End
  - HTML
    - structure
  - CSS
    - formatting
  - JavaScript
    - Client-side functionality
- Back-End (server side)
  - Old
    - ColdFusion

        - PHP
        - ASP
        - JSP
        - PERL
    - Newer
        - (4) Ruby
        - (3) Python
        - (2) Node.js
    - Newest
        - (1) Go

## About Go

- Official website
    - https://golang.org/
    - **https://golang.org/doc/faq#creating_a_new_language**
- Credentials
    - Google
        - Multiple cores
    - Luminaries in computer science
        - Rob Pike, Ken Thompson, Robert Griesemer
        - Helped create C, Unix, UTF-8

## Introduction to Programming

### How Computers Work

- https://play.golang.org/
- Circuits / switches
- Coding schemes
  - ascii
  - utf-8
  - Others
- Numeral systems
  - Decimal
  - Binary
  - Hex

### Variables

- A variable stores a value
- A variable has an identifier
- Example
  - X := 42
  - https://play.golang.org/p/APYjIf7KpN
- Block of code, aka, code block
  - Area between curly braces, aka, braces
  - Example {

    This area

    }
- Short declaration operator

    - For declaring variable in a code block
    - Example
      X := 42
- Var keyword
    - Examples
      var x int
      var y string
      var z bool
    - Assigns the zero value to the variable
        - https://golang.org/ref/spec#The_zero_value
    - Examples
      var x int = 42
      var y string = “Todd”
      var z bool = True
- Idiomatic code
    - Code that conforms to Go’s programming standards

## Hands-On

DECLARE & ASSIGN (INITIALIZE) TWO VARIABLES
print their sum

DECLARE YOUR NAME
print your name

Use the short declaration operator

## Hands On Solutions

- DECLARE & ASSIGN (INITIALIZE) TWO VARIABLES
    - Print their sum
    - https://play.golang.org/p/yLKZag1h1q

- DECLARE YOUR NAME
    - Print your name
    - https://play.golang.org/p/UJ5Agz4eQM

## Documentation - Standard Library

- Modularizing code
    - DRY - do not repeat yourself
    - Put code into
        - Functions
    - Put functions into
        - groups
- Strings
    - Double quotes
    - Backticks
        - Raw string literal
        - https://play.golang.org/p/NuS9wT-rRy
- Standard library
    - Strings package
    - Fmt package
- Two places to see documentation
    - Golang.org
        - Standard library
    - Godoc.org

        - Standard library
        - Third-party packages
        - Better formatting

## Value Types

- https://golang.org/ref/spec#Types
- Numeric
    - Talked about
        - Int
        - Uint
        - Rune
        - Byte
        - float64
    - In practice, most of the time just use
        - Int
        - float64
- String
- boolean

## Functions

https://play.golang.org/p/z4yvxT-GaD

- Func syntax
  ```
  func (receiver) identifier(parameters) (returns) {
  	// code goes here
  }
  ```
- Identifier naming conventions
    - short, concise, evocative
    - Camel case

        - Not underscores
        - *"... convention in Go is to use MixedCaps or mixedCaps rather than underscores to write multiword names."*
    - https://golang.org/doc/effective_go.html#names
    - Goal of Go code: readable code
- Software engineering
    - An engineering field
        - Requires precision
    - There's a language to talk about the language
        - Using precision in the words we use to talk about the language is important
- Terminology
    - Declare, Assign, Initialize
    - Parameter, Argument
- Hands-On

    - create a function that allows you to multiply two arguments which are both of type int and returns their result as an int

    - create a function that takes a string as an argument and then returns a string which does this: return ("Hello, " + argumentPassedInOfTypeString + "!")

## Hands-On Solutions

- create a function that allows you to multiply two arguments which are both of type int and returns their result as an int
  https://play.golang.org/p/j2sYxtlhOz

- create a function that takes a string as an argument and then returns a string which does this: return ("Hello, " + argumentPassedInOfTypeString + "!")
  https://play.golang.org/p/SrBYkYz1Ea

## Learning Command Line Interface (CLI) Basics

- CLI vs GUI
- CLI Terminology
    - POSIX (Unix, Linux, Mac)
        - Terminal, Bash, Shell
    - DOS (Windows)
        - Command prompt
- Terminal commands
    - ls
        - List
        - Lists everything in the current directory where you’re located at that moment
    - ls -la
        - List list-all
        - Lists everything with more information
    - pwd
        - Print Working
        - Directory
        - Shows the current directory where you're located
    - command + k
        - Clears terminal screen
    - clear
        - Clears terminal screen
    - cd
        - Change directory
        - Example
            - cd Documents
        - tab
            - Allows auto-completion matching anything you have started to type
            - Example

            - cd Doc (then hit tab) and it becomes cd Documents
    - cd ../
        - Moves up a directory
        - Examples
            - cd ../nanonan
            - cd
    - cd
        -
    - mkdir
        - Makes a directory (aka folder)
    - rm
        - rm -rf
            - Can also use that command
    - nano <file-name>
        - Creates a file, or if the file exists, opens the file in a text editor
        - On windows use the command ‘notepad’
    - env
        - Shows environment variables

## Windows Users - Installing Github Desktop and Git Shell

- [https://git-scm.com/](https://git-scm.com/)
    - Download and install for windows
        - This will work better than github desktop
            - And you might also need this on your machine
    - In the install process, make sure this is checked

- Git bash will allow us to use **POSIX CLI** commands on windows

## Hands-On Terminal Exercises

- Make a directory
- cd into that directory
- Make a file using nano
    - Save the file as “my-goals.txt”
- Remove that file
- Remove the directory

## Setting Up Your Go Workspace & Environment

- Install go
    - https://golang.org/dl/
- Go Workspace
    - Just create one
        - Create these folders
            - You can call the top folder anything, but I like ‘goworkspace’
                - goworkspace
                    - bin
                        - go install
                    - pkg
                        - precompiled packages; they will have a .a extension as in ‘**archive**’
                    - src
                        - source code
                        - Example folder structure
                            - github.com/github-username/repo-name
    - This convention allows
        - Name-spacing
        - Package management
            - go get <path to pkg on internet>

- Environment variables
    - GOPATH
        - Points to your goworkspace
    - GOROOTec
        - Points to your sdk
    - Set a path variable to the following
        - goworkspace/bin
- Setting environment variables MAC
    - At terminal,
        - cd
        - nano .bash_profile

          ```
          # for G O programming
          export GOROOT="/usr/local/go"
          export GOPATH="$HOME/Documents/goworkspace"
          export PATH="$HOME/Documents/goworkspace/bin:$PATH"
          export PATH="/usr/local/go/bin:$PATH"
          ```
        - ctrl + x
            - exits nano
        - Y
            - saves to file
        - close terminal & restart terminal
    - At terminal,
        - go env
        - echo $PATH
- Set environment variables WINDOWS
    - in windows explorer, right click this pc / properties
        - system properties
        - advanced system settings
        - environment variables go
            - under “System variables”
                - NEW

            - name: GOPATH
            - value: <path to your go workspace>
    - under “System variables”
        - choose PATH / edit
            - add path variable for
              C:\<folders to … your ...>\goworkspace\bin;
              C:\Go\bin;
                - this last one might already be in there, in which case don’t add it

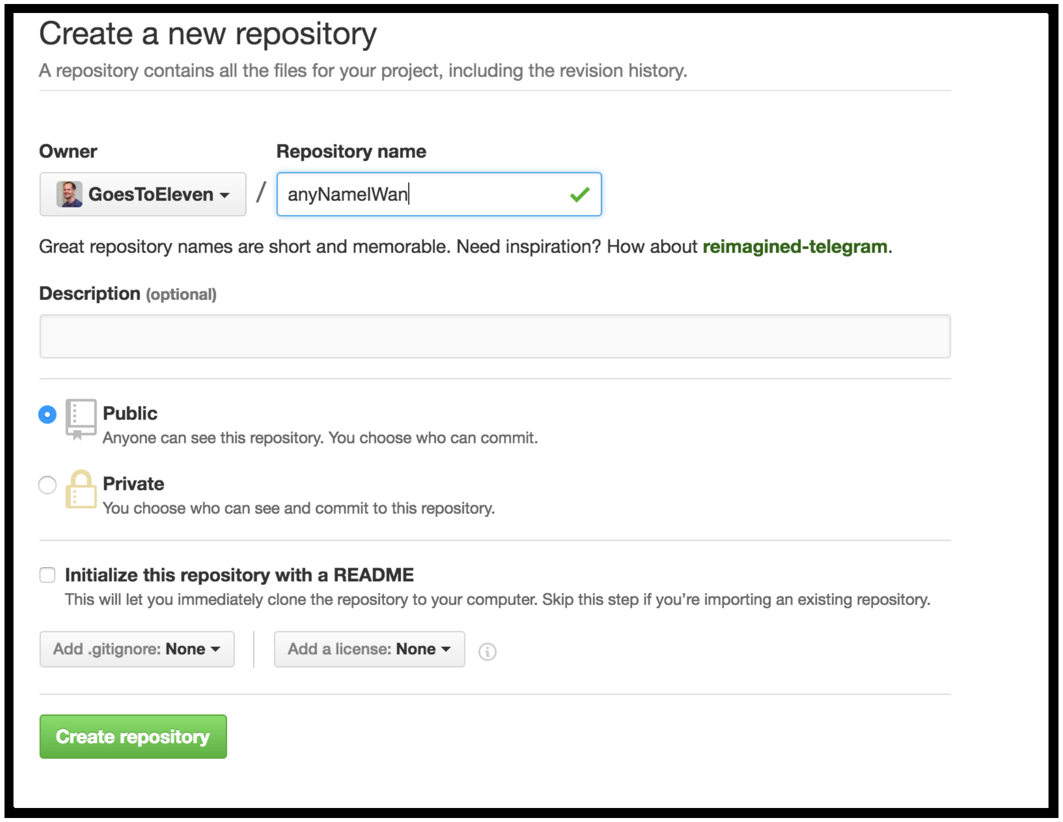
## Go Commands, go get, & Getting The Training Code From Github

- [https://golang.org/cmd/go/](https://golang.org/cmd/go/)
- at terminal
    - go get github.com/GoesToEleven/GolangTraining

## Creating A New Project - Github

- created a new repo

- with these settings



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: GoesToEleven / Repository name: anyNameIWan

Great repository names are short and memorable. Need inspiration? How about **reimagined-telegram**.

**Description** (optional)

(●) **Public** Anyone can see this repository. You choose who can commit.

( ) **Private** You choose who can see and commit to this repository.

[ ] **Initialize this repository with a README** This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None**

## Data Structures - Slice

- variable
  - holds a value of some type
- slice
  - holds lists of values of a certain type
  - https://play.golang.org/p/dH9cnwoBg5
    - cleaned up some: https://play.golang.org/p/WQsJTeEb6w

- hands on
  - create a slice of string using the var syntax; append to it your first name; append to it your last name; print it
  - create a slice of bool using the composite literal syntax; put the values true, false, true into it; print it

## Hands-On Solution

- https://play.golang.org/p/mmD5IB8tFg

## Slice, Array, & Performance

- t := make([]int, 0, 100)

## Map - key, value storage

- create a map using a composite literal
  - https://play.golang.org/p/tk0SiqgPtX
- make a map with make
  - https://play.golang.org/p/tSJQykkVKK
- hands on
  - create a map using a composite literal that stores author's name and one of their books; print one of the entries; print the whole map

## Hands-On Solution

- https://play.golang.org/p/mEz12JtSbL

## Struct

- composite data type, aka, aggregate data type
- one way
  - https://play.golang.org/p/I5cqIPUC9v
- another way
  - https://play.golang.org/p/INFTn0n40g
- another way
  - https://play.golang.org/p/2CbHyIv2IE
- aggregating together different data types
  - https://play.golang.org/p/3yomA8_9Xz
- inner type promotion
  - https://play.golang.org/p/nCa4tyK6cK
- attach method
  - https://play.golang.org/p/QmBw4yaDma
- receivers are like “this” in other programming languages; receivers are just another parameter which can be accessed like an argument when the func is called
  - https://play.golang.org/p/AmqtBU6WIs

## Range (loop) Over A Map or A Slice

- https://play.golang.org/p/iEpR9oM_us

## Interfaces

- types with methods
  - https://play.golang.org/p/RuavpWhypW
- https://golang.org/doc/effective_go.html#interfaces_and_types

    - Interfaces in Go provide a way to specify the behavior of an object: if something can do *this*, then it can be used *here*.
- interfaces and polymorphism
    - https://play.golang.org/p/XUe4UZ4tJk

## Web Templates

- get code
    - https://github.com/GoesToEleven/golang-web-dev
    - go get -u github.com/GoesToEleven/golang-web-dev
- templates
    - forms into which we insert data
- backtick
    - raw string literal
    - https://play.golang.org/p/sBt-UysJX2
- basic templating with strings and concatenation
    - https://play.golang.org/p/-o1IS-D67L

## HANDS ON

// HANDS ON 1
// create a struct that holds person fields
// create a struct that holds secret agent fields and embeds person type
// attach a method to person: pSpeak
// attach a method to secret agent: saSpeak
// create a variable of type person
// create a variable of type secret agent
// print a field from person

// run pSpeak attached to the variable of type person
// print a field from secret agent
// run saSpeak attached to the variable of type secret agent
// run pSpeak attached to the variable of type secret agent
SOLUTION: https://play.golang.org/p/RxrkCJw9Cd

// HANDS ON 2
// create a type square
// create a type circle
// attach a method to each that calculates area and returns it
// create a type shape which defines an interface as anything which has the area method
// create a func info which takes type shape and then prints the area
// create a value of type square
// create a value of type circle
// use func info to print the area of square
// use func info to print the area of circle
https://play.golang.org/p/1enChb7Kg5