Golang Notes

Sabrina Jiang

June 5, 2017

Contents

1 Basics

- Packages
  - Programs start running in package `main`
  - Can also import packages using the below syntax

    ```
    import (
        "fmt"
        "math/rand"
    )
    ```

  - Exported names are **capitalized** (e.g. `Pi` is exported from the package `math`)
- Functions
  - Basic Function Syntax

    ```
    func [functionName]([varOneName], [varTwoName] [varOneAndTwoType], [etc
        return [thing here]
    }
    ```

    * A return statement without arguments will return all named variables
- Variable Declaration
  - Variables can be declared without a type (e.g. `var c`)
  - Variables that are initialised must have a type (e.g. `var i int = 2`)
  - Variables can also be declared with the `:=` shorthand (e.g. `k := 3`)
    * Variables declared this way have their type inferred
    * e.g. `42` is an `int` while `3.142` is a `float64`
  - Constants cannot be declared with `:=`
  - Variables declared with types but no values are initialized with zero values (`0` for numeric, `false` for boolean, `""` for strings)
  - You can convert between types by using the type as a function (e.g. from `int` to `float64`, use `float64(i)`)

2 Flow Control

- For Loop Syntax

```
for [initializer] ; [condition] ; [post statement] {
    [code here]
}
```

  - Note the lack of parentheses around the components of the for loop
  - The init and post statements are optional (basically making this into a while loop)
  - A for loop without a post statement is an infinite loop

- If Syntax

```
if [statement] {
    [code]
} else {
    [more code]
}
```

- Switch Statement Syntax

```
switch [to be checked against] {
    case [case1]:
        [code execution]
    case [case2]:
        [code execution]
    default:
        [code default]
}
```

  - Once the code hits a case that succeeds it automatically breaks
  - A switch statement without a init is defaulted to be checked against `true`

- Defer
  - Arguments are evaluated immediately but the function is not called until after
  - Deferred functions are pushed onto a stack and executed in a **last-in-first-out**

3 More Data Types

- Pointers
  - A type `*T` is a pointer to the value of `T`
  - It's zero value is `nil`
  - The `&` operator generates a pointer to its operand

    ```
    i := 42
    p = &i
    ```

- Struct
  - Collection of fields
  - Constructed via the following

    ```
    type [name] struct {
      [varName] [varType]
      [varName] [varType]
    }
    ```

  - You can create a pointer to structs but do not need to dereference them in order to change values