# Organizing Programs

**Michael VanSickle**

@vansimke

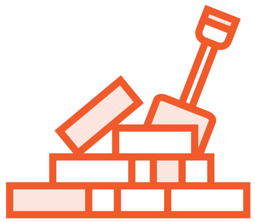
# Introduction

**Functions**

**Packages**

**Comments**

# Functions



**Function signatures**


**Parameters and arguments**

**Returning values**

# Function Signatures

```
func functionName(parameters)(return values){
  function body
}
```

# Parameters and Arguments

```
func functionName(parameters)(return values){
  function body
}
```

# Parameters and Arguments

```
func greet(name string) {
    fmt.Println(name)
}
```

# Parameters and Arguments

```
func greet(name1 string, name2 string) {
    fmt.Println(name1)
    fmt.Println(name2)
}

func greet(name1, name2 string) {
    fmt.Println(name1)
    fmt.Println(name2)
}
```

# Variadic Parameters

```
func greet(names ...string) {
    for _, n := range names {
        fmt.Println(n)
    }
}
```

**Variadic parameters**

- Received as slice
- Must be final parameter

# Passing Values and Pointers

```
func main() {
    name, otherName := “Name”, “Other name”
    fmt.Println(name)
    fmt.Println(otherName)
    myFunc(name, &otherName)
    fmt.Println(name)
    fmt.Println(otherName)
}

func myFunc(name string, otherName *string) {
    name = “New name”
    *otherName = “Other new name”
}
```

**Name**
**Other name**
**Name**
**Other new name**

Use pointers to share memory,
otherwise use values

# Return Values

```
func functionName(parameters)(return values){
  function body
}
```

# Returning Single Values

```
func main() {
    result := add(1, 2)
    fmt.Println(result)
}

func add(l, r int) int {
    return l + r
}
```

# Returning Multiple Values

```
func main() {
     result, ok := divide(1, 2)
     if ok {
          fmt.Println(result)
     }
}

func divide(l, r int) (int, bool) {
     if r == 0 {
          return 0, false
     }
     return l/r, true
}
```

# Named Return Values

```
func main() {
    result, ok := divide(1, 2)
    if ok {
        fmt.Println(result)
    }
}

func divide(l, r int) (result int, ok bool) {
    if r == 0 {
        return                          // 0, false
    }
    result = l/r
    ok = true
    return
    // optional: return l/r, true
}
```

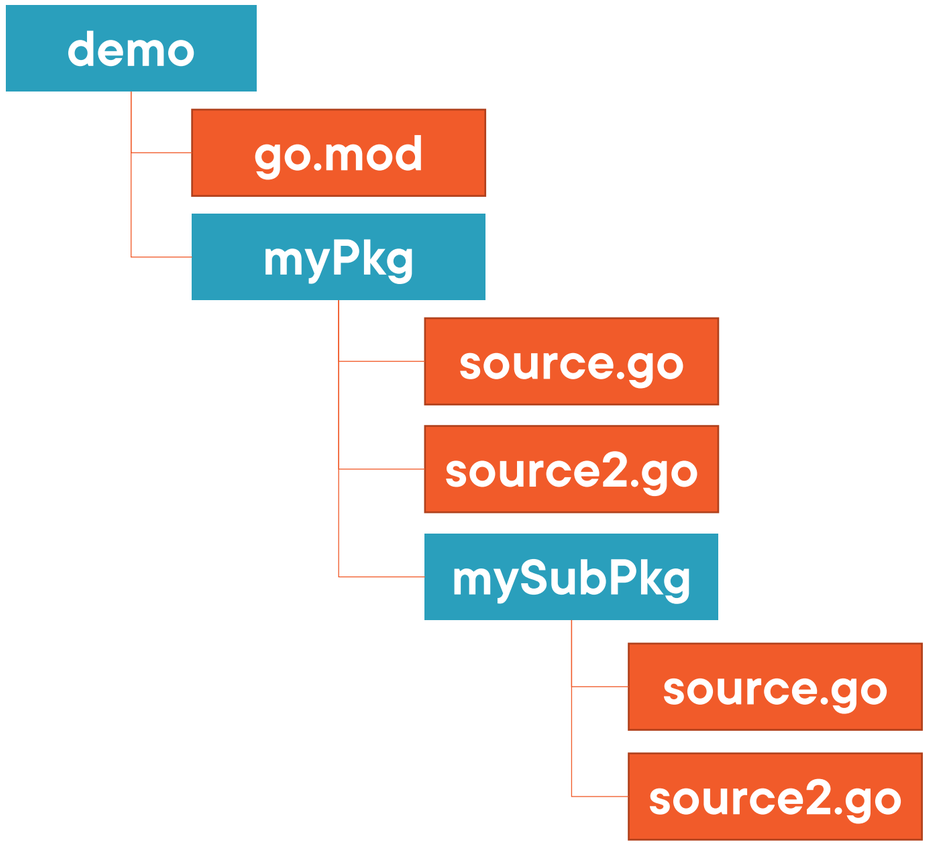
**rarely used**

# Demo

**Functions**

**refactor course demo to use functions to improve structure**

# Package Architecture




**Directory within a module**

**Contains at least one source file**

**All members are visible to other package members**

```
package user

import “strings”

var currentUsers []*User

const MaxUsers = 100

func GetByID(id int) (User, bool) {}
```

- package declaration
- import statement
- variable
- constant
- function

```
package user


type User struct {
    ID          int
    Username    string
    password    string
}
func NewUser() *User {}
const maxUsers = 100
```

◀ package identifier

◀ public struct

◀ public field

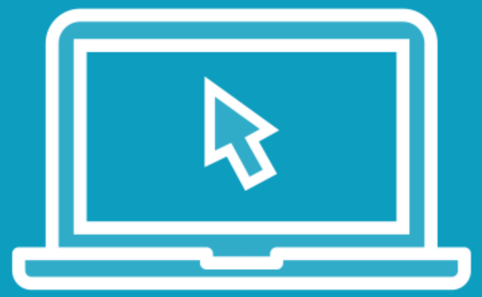
◀ public field

◀ package-level field

◀ public function

◀ package-level constant

# Demo



**Packages**

**refactor course demo to use packages to improve structure**

# Documentation is for users.

**Go Proverbs - https://go-proverbs.github.io/**

# Comments

```
// this is a single-line comment

var i int // single line comments can be added at the end of a line

/*
this is a multi line comment
*/
```

# Documenting Packages

```
// Package user provides functionality for managing
// users and their access rights.
package user

import “strings”

var currentUsers []*User
// MaxUsers controls how many users the system can handle at once.
const MaxUsers = 100

// GetById searches for users by their employee number.
func GetByID(id int) (User, bool) {}
```

# Demo

**Commentary**

**Show comments in std lib, focus on short comment statement**

**https://go.dev/doc/comment**

# Summary

**Functions**

**Packages**

**Comments**