# Week 3, Go

Azamat Serek, PhD, Assist.Prof.

# Packages

A package is made up of Go files that live in the same directory and have the same package statement at the beginning. You can include additional functionality from packages to make your programs more sophisticated. Some packages are available through the Go Standard Library and are therefore installed with your Go installation. Others can be installed with Go's go get command. You can also build your own Go packages by creating Go files in the same directory across which you want to share code by using the necessary package statement.

# Package under the organization

Before we create a new package, we need to be in our Go workspace. This is typically under our `gopath`. For the example, in this tutorial we will call the package `greet`. To do this, we've created a directory called `greet` in our `gopath` under our project space. If our organization were `gopherguides`, and we wanted to create the `greet` package under the organization while using Github as our code repository, then our directory would look like this:

```
└── $GOPATH
    └── src
        └── github.com
            └── gopherguides
```

# Greet directory

The `greet` directory is within the `gopherguides` directory:

```
└── $GOPATH
    └── src
        └── github.com
            └── gopherguides
                └── greet
```

# Entry point of the package

Finally, we can add the first file in our directory. It is considered common practice that the `primary` or `entry point` file in a package is named after the name of the directory. In this case, we would create a file called `greet.go` inside the `greet` directory:

```
└── $GOPATH
    └── src
        └── github.com
            └── gopherguides
                └── greet
                    └── greet.go
```

With the file created, we can begin to write our code that we want to reuse or share across projects. In this case, we will create a function called `Hello` that prints out `Hello World`.

# greet.go

Open your `greet.go` file in your text editor and add the following code:

greet.go

Copy

```
package greet

import "fmt"

func Hello() {
        fmt.Println("Hello, World!")
}
```

# Modules

In Go, a module is a collection of Go packages that are versioned together as a single unit. Each Go module is represented by a go.mod file, which contains information about the module, its dependencies, and version information. A package, on the other hand, is a collection of Go source files in the same directory that are compiled together.

# Create Module

1. **Create a New Directory:**

   Start by creating a new directory for your Go module. This directory will be the root of your project.

   bash Copy code

   ```
   mkdir mymodule
   cd mymodule
   ```

2. **Initialize the Module:**

   Run the following command to initialize a new Go module. This will create a `go.mod` file in your project's root directory.

   bash Copy code

   ```
   go mod init <module-path>
   ```

   Replace `<module-path>` with the import path you want to use for your module (e.g., `github.com/yourusername/mymodule`).

   Example:

   bash Copy code

   ```
   go mod init github.com/yourusername/mymodule
   ```

# Steps to create module

3. **Create Go Files:**

   Create your Go source files in the module directory. You can organize your code into different packages within the module.

4. **Write Code:**

   Write your Go code in the created files. For example, you might have multiple packages with functions or types in different files.

# Packages

## Creating Packages:

Go packages are collections of Go source files in the same directory that are compiled together. Here's how you can create packages:

1. **Organize Your Code:**
   Create subdirectories within your module directory to organize your code into packages.

plaintext Copy code

```
mymodule/
  |- go.mod
  |- main.go
  |- pkg/
     |- package1/
        |- file1.go
        |- file2.go
     |- package2/
        |- file3.go
     ...
```

# Package name

2. **Define Package Name:**

   Make sure to declare the package name at the beginning of each Go source file.

go Copy code

```go
// file1.go
package package1

// file2.go
package package1
```

go Copy code

```go
// file3.go
package package2
```

# Importing packages

3. **Importing Packages:**

   You can import packages from the same module or external modules in your Go files.

go Copy code

```go
// main.go
package main

import (
    "fmt"
    "github.com/yourusername/mymodule/pkg/package1"
    "github.com/yourusername/mymodule/pkg/package2"
)

func main() {
    fmt.Println(package1.Function1())
    fmt.Println(package2.Function2())
}
```

# Build and run

### Building and Running:

After organizing your code into packages and modules, you can build and run your Go application using the following commands:

bash Copy code

```
go build          # Build the application
./mymodule        # Run the executable
```

Remember to replace `mymodule` with the actual name of your module.

This basic structure can be expanded upon as your project grows. Additionally, tools like `go test` can be used for testing packages, and the `go install` command can be used to install your module globally.

## 1. Create the First Module and Package:

1. Create a directory for the first module:

bash Copy code

```
mkdir mathapp
cd mathapp
```

2. Initialize the module:

bash Copy code

```
go mod init github.com/yourusername/mathapp
```

3. Create the first package named `**arithmetic**`:

bash Copy code

```
mkdir arithmetic
```

4. Inside the `arithmetic` package, create two files:

- `add.go`:

go Copy code

```go
// arithmetic/add.go
package arithmetic

func Add(a, b int) int {
    return a + b
}
```

- `subtract.go`:

go Copy code

```go
// arithmetic/subtract.go
package arithmetic

func Subtract(a, b int) int {
    return a - b
}
```

# 2nd module

## 2. Create the Second Module and Package:

1. Create a directory for the second module:

bash Copy code

```
cd ..
mkdir geometryapp
cd geometryapp
```

2. Initialize the module:

bash Copy code

```
go mod init github.com/yourusername/geometryapp
```

3. Create the second package named `geometry`:

bash Copy code

```
mkdir geometry
```

# Files in package

4. Inside the `geometry` package, create two files:
   - `area.go`:

go Copy code

```go
// geometry/area.go
package geometry

func RectangleArea(length, width float64) float64 {
    return length * width
}
```

   - `circumference.go`:

go Copy code

```go
// geometry/circumference.go
package geometry

import "math"

func CircleCircumference(radius float64) float64 {
    return 2 * math.Pi * radius
}
```

# Creating folder for package

### 3. Use the Packages in a Main Program:

1. Create a main program that uses packages from both modules:

bash

Copy code

```
cd ..
mkdir mainapp
cd mainapp
```

# Package main

2. Inside the `mainapp` directory, create a file named `main.go`:

go

Copy code

```go
// mainapp/main.go
package main

import (
    "fmt"
    "github.com/yourusername/mathapp/arithmetic"
    "github.com/yourusername/geometryapp/geometry"
)
```

# main

```
func main() {
    // Use the arithmetic package from the mathapp module
    sum := arithmetic.Add(5, 3)
    difference := arithmetic.Subtract(10, 4)

    fmt.Printf("Arithmetic Results:\nSum: %d\nDifference: %d\n\n", sum

    // Use the geometry package from the geometryapp module
    rectangleArea := geometry.RectangleArea(4.0, 6.0)
    circleCircumference := geometry.CircleCircumference(3.0)

    fmt.Printf("Geometry Results:\nRectangle Area: %.2f\nCircle Circum
}
```

## 4. Build and Run:

1. Navigate to the `mainapp` directory and build the executable:

bash Copy code

```
go build
```

2. Run the executable:

bash Copy code

```
./mainapp
```

This example demonstrates the use of two modules (`mathapp` and `geometryapp`) and two packages (`arithmetic` and `geometry`). The main program in the `mainapp` module imports and uses functions from both packages.

# Practice exercise

Create 3 modules of your choice and at least 1 package in each module. The third module should call the functions of each package of each of the modules (1 and 2).

# References

https://go.dev/doc/tutorial/database-access