Getting Started with Go

# Resources

|  |  |
| --- | --- |
| <https://go.dev/doc/effective_go> | <https://go.dev/doc/> |
| <https://gobyexample.com/> | <https://gowebexamples.com/> |
| <https://go.dev/ref/spec> | <https://go.dev/ref/mod> |
| <https://go.dev/blog/> | <https://go.dev/blog/using-go-modules> |
| <https://pkg.go.dev/std> | <https://go.dev/ref/mem> |
|  |  |
|  |  |
|  |  |

# Packages

A package in Go is essentially a named collection of one or more related .go files. In Go, the primary purpose of packages is to help you **isolate and reuse code**.

Every .go file that you write should begin with a package {name} statement which indicates the name of the package that the file is a part of.

It's important to explain that code in a package can access and use **all types, constants, variables and functions within that package** **— even if they are declared in a different .go file**.

To run the codes:

go run \*.go or go run “list of all go files (I think the main package ones)”

## The main Package

* In Go, main is actually a special package name which indicates that the package contains **the code for an *executable application***. That is, it indicates that the package contains code that can be built into a binary and run.
* You’ll also get an error if you try to run a go file that is not in the main package. So, I guess it’s safe to say that you need a package called main.
* Any package with the name main must also contain a main() function **somewhere in the package** which **acts as the entry point for the program**. If it doesn't, and you try to run it, you will get an error.
* It’s conventional to put it in a file named main.go but you don’t have to.

## Importing from other Packages

individual .go files can import and use **exported types**, **constants**, **variables and functions** from other packages — **including** the packages in the Go standard library.

### Exported vs Unexported

something in Go code is exported if its name starts with a capital letter. Otherwise, it is unexported.

Unexported items are private to the package that they are declared in.

Exported items are public and visible to any package that imports the containing package.

Do not make your code exported unless you have a reason to.

### Importing and Using Go Standard Library Packages

* When importing a package **from the standard library** you need to use the full path to the package **in the standard library tree**, not just the name of the package.
* When using them as accessor to methods and variables, etc. though you just mention the last part of the path which is the name of the package.

package main

import (

“fmt”

**"math/rand"**

)

func randomNumber() int {

fmt.Printf(“something”)

return **rand**.Intn(100)

}

### Unused and Missing Imports

If you import a package but don't actually use it in your code, it will result in a compile-time error.

Similarly, you'll also get a compile-time error if a package is referenced in your code but not imported.

*There is a tool called* ***goimports*** *that automatically removes unused imports and imports referenced functions. I think VSCode uses it so you have them handled upon save. However, you must be careful when it imports packages like rand that there are packages with the same name as them but in a different path :* crypto/rand & math/rand

# Modules

if you want to import and use a third-party package — or structure your code so it's split into multiple packages — then you first need to turn your code into a Go module.

A module is... a tree of Go source files with a go.mod file in the tree's root directory.