Tips for Importing Packages

Useful tips and tricks for programming in Go.

WE'LL COVER THE FOLLOWING

This section will grow over time but the main goal is to share some tricks experienced developers discovered over time. Hopefully this tips will get new users more productive faster.

140 char tips

- leave your object oriented brain at home. Embrace the interface. @mikegehard
- Learn to do things the Go way, don't try to force your language idioms into Go. @DrNic
- It's better to over do it with using interfaces than use too few of them. @evanphx
- Embrace the language: simplicity, concurrency, and composition. @francesc
- read all the awesome docs that they have on golang.org. @vbatts
- always use `gofmt`. @darkhelmetlive
- read a lot of source code. @DrNic
- Learn and become familiar with tools and utilities, or create your own! They are as vital to your success as knowing the language. @coreyprak

They are as vital to your success as knowing the language. @coreyprak

Alternate Ways to Import Packages

There are a few other ways of importing packages. We'll use the `fmt` package in the following examples:

- `import format "fmt"` - Creates an alias of `fmt`. Preceed all `fmt` package content with `format.` instead of `fmt.`.
- `import . "fmt"` - Allows content of the package to be accessed directly, without the need for it to be preceded with `fmt`.
- `import _ "fmt"` - Suppresses compiler warnings related to `fmt` if it is not being used, and executes initialization functions if there are any. The remainder of `fmt` is inaccessible.

See this blog post for more detailed information.

goimports

Goimports is a tool that updates your Go import lines, adding missing ones and removing unreferenced ones.

It acts the same as gofmt (drop-in replacement) but in addition to code formatting, also fixes imports.

Organization

Go is a pretty easy programming language to learn but the hardest thing for developers at first is how to organize their code. Rails became popular for many reasons and scaffolding was one of them. It gave new developers clear directions and places to put their code and idioms to follow.

To some extent, Go does the same thing by providing developers with great tools like `go fmt` and by having a strict compiler that won't compile unused variables or unused import statements.

Custom Constructors

A question I often hear is when should I use custom constructors like `NewJob`. My answer is that in most cases you don't need to. However, whenever you need to set your value at initialization time and you have some sort of default

values, it’s a good candidate for a constructor. In the above example, adding a constructor makes a lot of sense so we can set a default logger.

Environment Variables

Key:	Value:
GOPATH	/go

```
package main

import (
	"log"
	"os"
)

type Job struct {
	Command string
	*log.Logger
}

func NewJob(command string) *Job {
	return &Job{command, log.New(os.Stdout, "Job: ", log.Ldate)}
}

func main() {
	NewJob("demo").Print("starting now...")
}
```

Breaking down code in packages

See this blog post on refactoring Go code, the first part talks about package organization.