Formatting the Code

This lesson highlights the importance of formatting in Go and a tool that performs formatting automatically.

WE'LL COVER THE FOLLOWING

Formatting in Go

Go's authors didn't want endless discussions about code-style for the Go-language. Specifically, the discussions that were indeed, a waste of precious development time for so many languages in the past. So, they made a tool: `go fmt` (or with more possibilities: `gofmt`). It is a pretty-printer that imposes the official, standard code formatting and styling on the Go source-code. It is also a syntax level rewriting tool, a simple form of refactoring. It is used rigorously in Go automatically and should be used by all Go-developers. Use `gofmt` on your Go-program before compiling or checking in to format it correctly to the idiomatic Go style.

Formatting code with `gofmt`

In most IDE's you can configure that saving the latest changes to the source-file also automatically formats the code with `gofmt`. For *indentation* of different levels in code, the rule is not strict: tabs or spaces can be used. A tab can be 4 or 8 spaces. In writing code in an editor or IDE, use tabs, don't insert spaces. On the command-line, you use it as follows:

```
go fmt program.go
```

This reformats program.go (without –w the changes are shown, but not saved). `go fmt *.go` works on all go-files; instead of source files, you can also specify packages. `gofmt folder1` works recursively on all **.go** files in the **folder1** directory and its subdirectories.

It can also be used for simple changes (refactoring) in a codebase by specifying a rule with the –r flag and a rule between `'` the rule has to be of the form:

```
pattern -> replacement
```

Look at some of the examples below:

```
gofmt -r '(a) -> a' -l *.go
```

This will replace all unnecessary doubled (( )) with () in all go-files in the current directory.

```
gofmt -r 'a[n:len(a)] -> a[n:]' -w *.go
```

This will remove all superfluous `len(a)` in such slice-declarations.

```
gofmt -r 'A.Func1(a,b) -> A.Func2(b,a)' -w *.go
```

This will replace `Func1` with `Func2` and swap the function's arguments.

For more info, visit this page.

The `gofmt` tool saves the time of users and makes the code readable. Now, it's time to learn how to build a program and run it locally on your machine.