Refactoring Go code

One of the advantages of Go is that refactoring Go code is easier
than in some other languages. This workshop will explore some
of ways that Go code can be refactoried manually and automatically.

Why do we refactor code?

- increasing code complexity
- changing requirements
- changing dependencies

Some ways that we might wish to refactor code:

- rename variables, types and methods
- change a function into a method
- pass an interface around instead of a static type
- add, remove or reorder arguments to a function
- move a package path

Exercises

1. Renaming. Let the compiler tell you what to do. Take some Go code,
   change a name somewhere and try to compile it.
   The compiler should say where there's an issue. Fix the code manually.

2. Methods and functions can often be interchangable.
   Take some method and factor it out into a function.
   Use gofmt to change all calls to it.

3. Sometimes, we want to be able to change some algorithm
   to work on an interface rather than a static type.
   Take a method and factor it out into a function that takes
   the receiver as an interface first argument. You'll need
   to define the interface type too, holding the methods
   of the type used by the new function.

4. Automatic renaming. The `gofmt` command can be used
   to automatically rename variable, function and method names.
   Note that this is type agnostic.
   Use `gofmt -r` to do a rename automatically.

5. The `gofmt` can also be used to add, remove and reorder
   arguments to functions. Try reordering the arguments to some
   function or method with gofmt.

6. The `govers` command can be used to update import paths in Go source.
   Although originally written for updating gopkg.in package versions,
   arbitrary packages can be changed by using the `-m` flag. Get govers with
   `go get github.com/rogpeppe/govers`. Let's pretend we're going to fix
   the `encoding/xml` package and change a bunch of code to use it. Fork
   the encoding/xml package into a place of your choice (just copy all
   the files in that package from the Go root source directory). Then find
   some place that uses encoding/xml (`golang.org/x/tools` is one example)
   and use govers to make all the packages use the new dependency.

7. Unwanted dependencies. Sometimes we'll find that a project has acquired an unwanted dependency and it's hard to find out why the dependency is required. The `showdeps` command can be used to investigate dependency requirements. It can be found at `github.com/rogpeppe/showdeps`. The `-why` flag shows all dependency routes between a set of packages and some dependency. Investigate some dependencies of some Go code with `showdeps -a`. Do you know why all those dependencies are there? Use the `-why` flag to find out why.

8. Splitting up a function. Sometimes a method or function will grow larger than is comfortable and you'll want to split it up. When there's variable mutation in many parts of the function, that can be awkward. It can be useful to create a type that's used only for the duration of that function, holding what were previously function-local variables. For example, it could be argued that `Decoder.unmarshal` in the stdlib's `encoding/xml` pacakge is too big. Split it up by creating a new type, moving some of the local variables into it, and moving parts of `Decoder.unmarshal` into methods on the new type.

9. Other tools. There are some other tools in `golang.org/x/tools` which are there for Go refactoring. I haven't found them that useful so far, but perhaps you might. Investigate them and see whether you can get them to do something useful for you. Specifically the following:

- `golang.org/x/tools/cmd/eg`
- `golang.org/x/tools/cmd/gomvpkg`
- `golang.org/x/tools/cmd/gorename`

10. The `go/ast`, `go/types` and other related packages provide the ability for Go code to parse and manipulate other Go code, and the `go/fmt` package makes it easy to print out well formatted code. Although it's unusual to need it, when making large scale changes across a big code base, this can be very useful. Write some code to parse some Go code, make some changes to the AST and print the result.