MORE ABOUT MODULES

Before moving on to learning more about writing functions and barely useful snippets of code, there are a few other miscellaneous bits of information that might be useful to you in the future that I'd like to discuss.

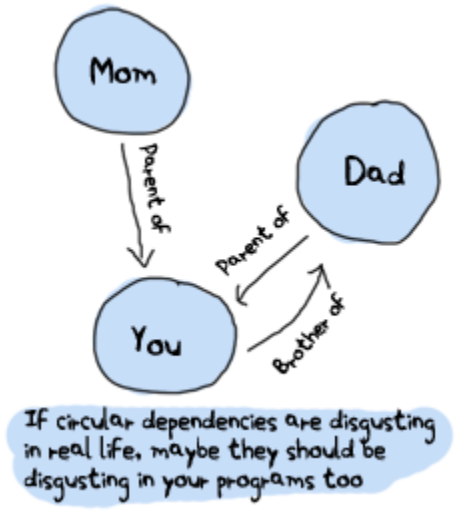
The first one concerns metadata about modules. I mentioned in the beginning of this chapter that module attributes are metadata describing the module itself. Where can we find this metadata when we don't have an access to the source? Well the compiler plays nice with us: when compiling a module, it will pick up most module attributes and store them (along with other information) in a `module_info/0` function. You can see the metadata of the `useless` module the following way:

```
9> useless:module_info().
[{exports,[{add,2},
{hello,0},
{greet_and_add_two,1},
{module_info,0},
{module_info,1}]},
{imports,[]},
{attributes,[{vsn,[174839656007867314473085021121413256129]
}]},
{compile,[{options,[]},
{version,"4.6.2"},
{time,{2009,9,9,22,15,50}},
{source,"/home/ferd/learn-you-some-erlang/useless.erl"}]}]
10> useless:module_info(attributes).
[{vsn,[174839656007867314473085021121413256129]}]
```

The snippet above also shows an additional function, `module_info/1` which will let you grab one specific piece of information. You can see exported functions, imported functions (none in this case!), attributes (this is where your custom metadata would go), and compile options and information. Had you decided to add `-author("An Erlang Champ").` to your module, it would

have ended up in the same section as `vsn`. There are limited uses to module attributes when it comes to production stuff, but they can be nice when doing little tricks to help yourself out: I'm using them in my testing script for this book to annotate functions for which unit tests could be better; the script looks up module attributes, finds the annotated functions and shows a warning about them.

Note: `vsn` is an automatically generated unique value differentiating each version of your code, excluding comments. It is used in code hot-loading (upgrading an application while it runs, without stopping it) and by some tools related to release handling. You can also specify a `vsn`value yourself if you want: just add `-vsn(VersionNumber)` to your module.



Another point that would be nice to approach regards general module design: avoid circular dependencies! A module *A* should not call a module *B* that also calls module *A*. Such dependencies usually end up making code maintenance difficult. In fact, depending on too many modules even if they're not in a circular dependency can make maintenance harder. The last thing you want is to wake up in the middle of the night only to find a maniac software engineer or computer scientist trying to gouge your eyes out because of terrible code you have written.
For similar reasons (maintenance and fear for your eyes), it is usually considered a good practice to regroup functions that have similar roles close together. Starting and stopping an application or creating and deleting a record in some database are examples of such a scenario.

Well, that's enough for the pedantic moralizations. How about we explore Erlang a little more?

Source :http://learnyousomeerlang.com/modules#what-are-modules