Good Haskell Style

All your submitted programming assignments should *emerge creatively* from the following style guidelines. Programming is...

- **engineering**: every field of engineering has a set of *best practices* that help in producing high-quality designs.

- **communication**: social conventions make it easier to communicate by allowing others to focus on the *content* rather than the *form* of your program.

- **an art form**: as every artist knows, constraints serve to enhance rather than quench creativity.

You may also refer to [https://github.com/tibbe/haskell-style-guide/blob/master/haskell-style.md](https://github.com/tibbe/haskell-style-guide/blob/master/haskell-style.md) which goes into much more specific detail about best practices for formatting Haskell code.

- **DO** use *camelCase* for function and variable names.

- **DO** use descriptive function names, which are as long as they need to be but no longer than they have to be. Good: `solveRemaining`. Bad: `slv`. Ugly: `solveAllTheCasesWhichWeHaven’tYetProcessed`.

- **DON’T** use tab characters. Haskell is layout-sensitive and tabs Mess Everything Up. I don’t care how you feel about tabs when coding in other languages. Just trust me on this one. Note this does not mean you need to hit space a zillion times to indent each line; your Favorite Editor ought to support auto-indentation using spaces instead of tabs. That is, you can use the Tab *key* on your keyboard and have your editor automatically insert space *characters* in your document.

- **DO** try to keep every line under 80 characters. This isn’t a hard and fast rule, but code that is line-wrapped by an editor looks horrible.

---

\(^1\)Which I mostly agree with.
DO give every top-level function a type signature. Type signatures enhance documentation, clarify thinking, and provide nesting sites for endangered bird species. Top-level type signatures also result in better error messages. With no type signatures, type errors tend to show up far from where the real problem is; explicit type signatures help localize type errors.

Locally defined functions and constants (part of a let expression or where clause) do not need type signatures. In fact, sometimes it can actually hurt: to use local type signatures which are polymorphic you need to enable a certain extension and jump through some hoops (ask for details if you are curious). If you need to add a polymorphic type signature to a local function (e.g. to help with debugging a type error) it’s usually a good idea to move it to the top level.

DO precede every top-level function by a comment explaining what it does (i.e. describe the inputs and the output).

DO use -Wall. Either pass -Wall to ghc on the command line, or (recommended) put

```
{-# OPTIONS_GHC -Wall #-}
```

at the top of your .hs file. All your submitted programs should compile with no warnings.

DO, as much as possible, break up your programs into small functions that do one thing, and compose them to create more complex functions.

DO try to make all your functions total. That is, they should give sensible results (and not crash) for every input.