Good Haskell Style

All your submitted programming assignments should *emerge creatively* from the following style guidelines. Programming is just as much social art form as it is engineering discipline, and as any artist knows, constraints serve to enhance rather than quench creativity.

These guidelines will be extended as the semester progresses.

- **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.
- **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.
- **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, but adding them doesn't hurt (in particular, the argument above about localizing type errors still applies).
- **DO** precede every top-level function by a comment explaining what it does.
- **DO** use `-Wall`. Either pass `-Wall` to `ghc` on the command line, or (easier) 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** make all your functions *total*. That is, they should give sensible results (and not crash) for every input.