WIKIBOOKS

Haskell/Indentation

Haskell relies on indentation to reduce the verbosity of your code. Despite some complexity in practice, there are really only a couple fundamental layout rules.[1]

The golden rule of indentation

Code which is part of some expression should be indented further in than the beginning of that expression (even if the expression is not the leftmost element of the line).

The easiest example is a 'let' binding group. The equations binding the variables are part of the 'let' expression, and so should be indented further in than the beginning of the binding group: the 'let' keyword. When you start the expression on a separate line, you only need to indent by one space (although more than one space is also acceptable and may be clearer).

```
let
 x = a
 y = b
```

You may also place the first clause alongside the 'let' as long as you indent the rest to line up:

wrong

```
let x = a
 y = b
```

wrong

```
let x = a
      y = b
```

right

```
let x = a
    y = b
```

This tends to trip up a lot of beginners: All *grouped* expressions must be exactly aligned. On the first line, Haskell counts everything to the left of the expression as indent, even though it is not whitespace.

Here are some more examples:

```
do
  foo
  bar
  baz

do foo
   bar
   baz

where x = a
      y = b

case x of
  p  -> foo
  p' -> baz
```

Note that with 'case' it is less common to place the first subsidiary expression on the same line as the 'case' keyword (although it would still be valid code). Hence, the subsidiary expressions in a case expression tend to be indented only one step further than the 'case' line. Also note how we lined up the arrows here: this is

purely aesthetic and is not counted as different layout; only *indentation* (i.e. whitespace beginning on the far-left edge) makes a difference to the interpretation of the layout.

Things get more complicated when the beginning of an expression is not at the start of a line. In this case, it's safe to just indent further than the line containing the expression's beginning. In the following example, `do` comes at the end of a line, so the subsequent parts of the expression simply need to be indented relative to the line that contains the `do`, not relative to the `do` itself.

```
myFunction firstArgument secondArgument = do
  foo
  bar
  baz
```

Here are some alternative layouts which all work:

```
myFunction firstArgument secondArgument =
  do foo
     bar
     baz

myFunction firstArgument secondArgument = do foo
                                              bar
                                              baz
myFunction firstArgument secondArgument =
  do
     foo
     bar
     baz
```

Explicit characters in place of indentation

Indentation is actually optional if you instead use semicolons and curly braces for grouping and separation, as in "one-dimensional" languages like C. Even though the consensus among Haskell programmers is that meaningful indentation leads to better-looking code, understanding how to convert from one style to the other can help understand the indentation rules. The entire layout process can be summed up in three translation rules (plus a fourth one that doesn't come up very often):

1. If you see one of the layout keywords, (`let`, `where`, `of`, `do`), insert an open curly brace (right before the stuff that follows it)
2. If you see something indented to the SAME level, insert a semicolon
3. If you see something indented LESS, insert a closing curly brace
4. If you see something unexpected in a list, like `where`, insert a closing brace before instead of a semicolon.

For instance, this definition...

```
foo :: Double -> Double
foo x =
    let s = sin x
        c = cos x
    in 2 * s * c
```

...can be rewritten without caring about the indentation rules as:

```
foo :: Double -> Double;
foo x = let {
  s = sin x;
```

```
   c = cos x;
   } in 2 * s * c
```

One circumstance in which explicit braces and semicolons can be convenient is when writing one-liners in GHCi:

```
Prelude> let foo :: Double -> Double; foo x = let { s = sin x; c = cos x } in 2 * s * c
```

Exercises

Rewrite this snippet from the Control Structures chapter using explicit braces and semicolons:

```
doGuessing num = do
  putStrLn "Enter your guess:"
  guess <- getLine
  case compare (read guess) num of
    LT -> do putStrLn "Too low!"
             doGuessing num
    GT -> do putStrLn "Too high!"
             doGuessing num
    EQ -> putStrLn "You Win!"
```

Layout in action

wrong	wrong	right	right

`do first thing`
`second thing`
`third thing`
	`do first thing`
	`second thing`
	`third thing`
		`do first thing`
		`second thing`
		`third thing`
			`do`
			`first thing`
			`second thing`
			`third thing`

Indent to the first

Due to the "golden rule of indentation" described above, a curly brace within a `do` block depends not on the `do` itself but the thing that immediately follows it. For example, this weird-looking block of code is totally acceptable:

```
        do
first thing
second thing
third thing
```

As a result, you could also write combined if/do combination like this:

Wrong	Right	Right

`if foo`
`then do first thing`
`second thing`
`third thing`
`else do something_else`
	`if foo`
	`then do first thing`
	`second thing`
	`third thing`
	`else do something_else`
		`if foo`
		`then do`
		`first thing`
		`second thing`
		`third thing`
		`else do`
		`something_else`

It isn't about the `do`, it's about lining up all the items that are at the same level within the `do`.

Thus, all of the following are acceptable:

```
main = do
  first thing
  second thing
```

or

```
main =
  do
    first thing
    second thing
```

or

```
main =
  do first thing
     second thing
```

Notes

1. See section 2.7 of The Haskell Report (lexemes) (http://www.haskell.org/onlinereport/lexemes.html#sect2.7) on layout.