A module is a collection of related functions, types, and typeclasses.

A program is a collection of modules where the main module loads up the other modules and then uses their functions.

Prelude

```
:m + Module1 Module2 ... ModuleN
```

Or, load a script that does imports of its own. They too will be available.

```
import Module (func1, func2, ...)

import Module hiding (func1, func2, ...)
```

You can import one on a qualified basis so only the fully qualified name of one (or both) of the functions is available.

```
import qualified Module
```

To make this more convenient the module can be renamed to something shorter.

```
import qualified Module as M
```

`http://haskell.org/hoogle`

A Haskell-specific search engine that lets you search by name, module name, or even type signature.

They are strict versions of their lazy counterparts. They avoid stack overflow errors that result from folding on large lists.

A value that is yet to be evaluated.

`concat`

`nub`

`and` and `or`

`span` is like `takeWhile` but returns a pair of lists, the second of which is the list of elements that would have been dropped by `takeWhile`.

`break` negates `takeWhile`'s predicate.

It takes a list and groups adjacent elements into sublists if they are equal.

If you sort before grouping you can find out how many times each element appears in the list.

`isPrefixOf`, `isSuffixOf`, `isInfixOf`

It separates a list into a pair of two lists, those satisfying a predicate and those failing it.

```
lines, unlines

words, unwords
```

Functions whose names start with "generic" that correct the historical error of using `Int` instead of `Num` as the type of a parameter.

Functions whose names end with "by" test with a provided equality function instead of using `==`.

`on`, used infix between `(==)` and the function that gets the value you want to use for comparison.

Using `fromList` on a list of 2-tuples where the first element is an `Ord`.

They are represented using trees.

`lookup` returns a `Maybe`

`member`

`insert/delete`

As trees, so they're ordered.

Use `fromList`.

Use `singleton`.

`size, null, empty`

`member`

`insert, delete`

`nub` is simplest, requires only `Eq`, but is potentially slow.

Converting to a set with `fromList` and then back with `toList` may be faster but requires `Ord`.

```
module MyModule
( funcToExport1
, funcToExport2
...
) where
...
```