Assessment

- Coursework (25%)
  - Formative—to help you learn.
  - A series of exercises, done as 'homework'
  - Lab (or Labs?) simply to help get started with Haskell and all its bits. Only if you need help.
- Exam (75%)
  - New format for 2017/18:
    Programming Question
    Multiple Choice—there will be 'dry-run' exercises on Blackboard.

Lab 0 : Lab plan & procedure

- Details of exercise released before lab session
  - resources
  - task
  - submission procedure & deadline
- Lab Class
  - Help with exercises, any anything else Haskell-related.
  - Attendance at lab class is *NOT* required.
  - ICTLab I has 40 seats, class-size is 110 approx.
  - Attend *only* if you need help!
- Don't forget to submit the final version of your work!
- Deadlines are important:
  - exercises may be linked: solution to lab n could be input to lab $n+1$.
- First lab: Thursday 5th October, 2pm, ICTLAB1
- Other labs will only occur if there is a need.

Live Demo!

Time to live dangerously !

Haskell: Syntactical Details

- Now time for a proper introduction to the *language* of Haskell.
- Official Reference: "Haskell 2010 Language Report"
  - Online:
    `http://www.haskell.org/onlinereport/haskell2010/`
- In this course we refer to sections of that report thus:
  - [*H2010* 3.4]
  - Haskell 2010 Language Report, Section 3.4

Haskell is Case-Sensitive [*H2010* 2.4]

For example, the following names are all different:

```
ab
aB
Ab
AB
```

Program Structure [*H2010* 1.1]

A Haskell script can be viewed as having four levels:

1. A Haskell program is a set of *modules*, that control namespaces and software re-use in large programs.
2. A module consists of a collection of *declarations*, defining ordinary values, datatypes, type classes, and fixity information.
3. Next are *expressions*, that denote values and have static types.
4. At the bottom level is the *lexical structure*, capturing the concrete representation of programs in text files.

(We focus on the bottom three for now).

Comments [*H2010* 2.3]

A Haskell script has two kinds of comments:

1. End-of-line comments, starting with `--`.
2. Nested Comments, started with `{-` and ending with `-}`

Example, where comments are in red.

```
myfun x -- end-of-line, but -} won't end it
 = let
     y = 2 {- nested, but -- ignored here -} ; z = 3
     {-
     a = 4 {- was 42 but I changed my mind -}
     b = 5
     -}
   in y + z * x
```

Namespaces [*H2010* 1.4]

- Six kinds of names in Haskell:
  1. *Variables*, denoting values;
  2. *(Data-)Constructors*, denoting values;
  3. *Type-variables*, denoting types;
  4. *Type-constructors*, denoting 'type-builders';
  5. *Type-classes*, denoting groups of 'similar' types;
  6. *Module-names*, denoting program modules.
- Two constraints (only) on naming:
  - *Variables* (1) and *Type-variables* (3) begin with lowercase letters or underscore,
    Other names (2,4,5,6) begin with uppercase letters.
  - An identifier cannot denote both a *Type-constructor* (4) and *Type-class* (5) in the same scope.
- So the name `Thing` (*e.g.*) can denote a module, data-constructor, and either a class or type-constructor in a single scope.

Notational Conventions [*H2010* 2.1]

- The report uses the following notation for syntax:

[syn]	optional occurence of *syn*
{syn}	zero or more repetitions of *syn*
(syn)	grouping
$syn_1 \vert syn_2$	choice between alternatives
$syn_{\langle syn' \rangle}$	difference—elements generated by *syn*, except those generated by syn'
`fibonacci`	terminal syntax in `typewriter font`

- It uses BNF-like syntax, with productions of the form:

$$nonterm \rightarrow alt_1 \vert alt_2 \vert \ldots \vert alt_n$$

"*nonterm* is either an alt_1 or alt_2 or ..."

- The trick is distinguishing | (alternative separator) from `|` , the vertical bar character (and similarly for characters `{}[]()`).

Character Types (I) [*H2010* 2.2]

The characters can be grouped as follows:

- *special* : `( ) , ; [ ] ‘ { }`
- *whitechar* → *newline*|*vertab*|*space*|*tab*
- *small* → `a`|`b`|...|`z`|`_`
- *large* → `A`|`B`|...|`Z`
- *digit* → `0`|`1`|...|`9`
- *symbol* : `! # % & * + . / < = > ? @ \ ^ | - ~`
- the following characters are not explicitly grouped- `: " ’`

(There is also stuff regarding Unicode characters (beyond ASCII) that we shall ignore—so the above is not exactly as shown in [*H2010* 2.2]).

Lexemes (I) [*H2010* 2.4]

The term "lexeme" refers to a single basic "word" in the language.

- *Variable Identifiers* (*varid*) start with lowercase and continue with letters, numbers, underscore and single-quote.
  `x x’ a123 myGUI _HASH very_long_Ident_indeed’’`
- *Constructor Identifiers* (*conid*) start with uppercase letters and continue with letters, numbers, underscore and single-quote.
  `T Tree Tree’ My_New_Datatype Variant123`
- *Variable Operators* (*varsym*) start with any symbol, and continue with symbols and the colon.
  `<+> |:| ++ + - ==> == && #!#`
- *Constructor Operators* (*consym*) start with a colon and continue with symbols and the colon.
  `:+: :~ :=== :$%&`

Identifiers (*varid*, *conid*) are usually prefix, whilst operators (*varsym*,*consym*) are usually infix.

Lexemes (II) [*H2010* 2.4]

- *Reserved Identifiers* (*reservedid*):
  `case class data default deriving do else foreign if import in infix infixl infixr instance let module newtype of then type where _`
- *Reserved Operators* (*reservedop*):
  `.. : :: = \ | <- -> @ ~ =>`

Literals [*H2010* 2.5,2.6]

We give a simplified introduction to literals (actual basic values)

- *Integers* (*integer*) are sequences of digits
  Examples: `0 123`
- *Floating-Point* (*float*) has the same syntax as found in mainstream programming languages. `0.0 1.2e3 1.4e-45`
- *Characters* (*char*) are enclosed in single quotes and can be escaped using backslash in standard ways.
  `'a' '$' '\'' '"' '\64' '\n'`
- *Strings* (*string*) are enclosed in double quotes and can also be escaped using backslash in standard ways.
  `"Hello World" "I 'like' you"`
  `"\" is a dbl-quote" "line1\nline2"`