Install You a Haskell for Great Good!
Informatics 1 – Introduction to Computation
Functional Programming Tutorial 1

Week 1 (16-20 Sep.)

Welcome

Welcome to your first functional programming tutorial! This document will explain how to get started writing Haskell. *Please go through the entire worksheet in advance of the tutorial; answer the questions when needed or take some personal notes.*

The main purpose of the present tutorial is to familiarize yourselves with writing and using Haskell. You will be shown how you can use your favorite text editor to write a Haskell program and run it with the interactive Haskell interpreter GHCi. 'GHC' stands for 'Glasgow Haskell Compiler' or 'The Glorious Glasgow Haskell Compilation System'.

The tutorial consists of the following parts:

0. **Homework: Quiz** Read the questions beforehand and bring your answers to the tutorial.
1. **Tutorial Activities** Short group activities for getting-to-know each other.
2. **Install you a Haskell** In the second part you will set up the system and get to know the basic tools for programming.
3. **Getting Started** The third part consists of some simple exercises where you will write some arithmetic functions in Haskell. Read this beforehand. You will do the exercises during this tutorial.

Attendance at tutorials is obligatory; please send email to lambrose@exseed.ed.ac.uk if you cannot join your assigned tutorial.

0 Homework: Quiz

Bring your answers to the tutorial.

1. What is the file extension of a Haskell Script?
   (a) `.haskell`
   (b) `.hs`
   (c) `.txt`
2. What does this "`a b c d e`" set of identifiers represent in Haskell?
   (a) a list of characters.
   (b) a function call.
   (c) it's not a valid expression.
3. The type signature "`prefix :: String -> Int -> String`" means that:
   (a) the function `prefix` takes a string and an integer and a string as parameters.
   (b) the function `prefix` takes a string and an integer as parameters and returns a string.
4. The body of a function is:
   (a) a declaration.
   (b) an expression.
   (c) neither of the above.
5. When defining a function:
   (a) you could also provide the type signature.
   (b) you have to provide the type signature.
6. A function "`add :: Int -> Int -> Int`" can be called as (there are two correct answers):
   (a) `add 21 21`
   (b) `21` `add` `21`
   (c) `add(21, 21)`

1 Tutorial Activities

This is an introductory tutorial and its main goal is to help each of you install Haskell on your computer, consequently the group activities will be minimized — this time. So make sure that you don't spend more than 20min. in this section. Please also note that you are being encouraged to work together and help each other throughout the whole tutorial.

Please, take the first 5min. of the tutorial to introduce yourselves to your new group.

Exercise 1

3 × 2min. For each pair of the 1-6 Quiz questions, i.e. `[[1,2], [3,4], [5,6]]` split your group 2-2-2 or 2-3 (using a different split for each question) and compare your answers. Try to convince each other to agree on an answer.

5min. As a group, discuss your answers and check whether you can all agree on your final answer to all questions. If you need help raise your hand.

2 Install you a Haskell

Open a terminal and type "`ghci --version`" to see if you have "`ghci`" already installed on your computer. If not, please follow the link below in order to install the Haskell Platform by picking the correct version (Windows, OS X for Mac, Linux) and choosing the minimal installation:

https://www.haskell.org/downloads#platform

Once you have installed "`ghci`" install the `QuickCheck` package by running in your terminal:

```
$ cabal update
$ cabal install QuickCheck
```

You can use Haskell with any text editor as shown in this short video, which will be also demonstrated during the tutorial.

We will begin by using the Haskell REPL (read-eval-print-loop). This interactive environment is usually provided by GHCi, the interactive Haskell compiler/interpreter. At any time you can type "`:help`" at the prompt to see all the available commands.

Exercise 2

Start the Haskell REPL by just typing "`ghci`" in your terminal.

(a) Type "`3 + 4`" at the prompt. What does it say?

(b) Try "`3 + 4 * 5`" and "`(3 + 4) * 5`". Does arithmetic in Haskell work as expected?

(c) Find the length of a string by typing "`length "This is a string."`"

(d) Reverse the previous string using the... "`reverse`" function.

3 Getting Started

If the person sitting next to you is having difficulty, help them out!

Download the file `tutorial1.zip` from the course Learn page. and unpack it. It should contain a folder which in turn contains the files `tutorial1.hs` and `PicturesSVG.hs`. Use your favorite editor to open the `tutorial1.hs` file.

Below the introductory comments and the phrase `import Test.QuickCheck`, which loads the QuickCheck library that we will use later, you will find the type signature and an implementation of the `double` function.

Open your terminal and "`cd`" to the folder you have unzipped the exercises, next start the REPL in your terminal by typing "`ghci`".

Exercise 3

(a) Load the file `tutorial1.hs` into the REPL by using "`:load tutorial1`".

(b) Part of the definition (the line `double x = x + x`) is incorrectly indented: it should be vertically aligned with its type signature (the line above). Edit this line to correct the indentation.

(c) Save and reload the corrected file as shown in the video.

(d) Use the REPL to display

  i. the value of `double 21`

  ii. the type of `double`, by using the command "`:type functionName`"

  iii. the type of `double 21`

(e) What happens if you ask the REPL to evaluate `double "three"`?

(f) Complete the definition of `square :: Int -> Int` in `tutorial1.hs` so it computes the square of a number (you should replace the word "`undefined`"). Reload the file and test your definition.

Pythagorean Triples

Introduction — Read in advance Pythagoras was a Greek mystic who lived from around 570 to 490 BC. He is known to generations of schoolchildren as the discoverer of the relationship between the sides of a right-angled triangle. There is little evidence, however, that Pythagoras was a geometer at all. Early references to Pythagoras make no mention of his putative mathematical achievements, but refer instead to his pronouncements on dietary matters (he prohibited his followers from eating beans) or his less cerebral achievements such as biting a snake to death.

Whether or not Pythagoras had anything to do with the discovery of the theorem that bears his name, it was evidently known in antiquity. A stone tablet from Mesopotamia which predates Pythagoras by 1000 years, "Plimpton 322", appears to contain part of a list of "Pythagorean triples": positive integers corresponding to the lengths of the sides of a right-angled triangle. Back with the Greeks, Euclid (325 – 265BC) described a method for generating Pythagorean triples in his famous treatise *The Elements*.

In this part of the exercise we'll be taking a more modern approach to the ancient problem, using Haskell to generate and verify Pythagorean triples.

First, a formal definition: a *Pythagorean triple* is a set of three integers (a, b, c) which satisfy the equation $a^2+b^2 = c^2$. For example, $(3, 4, 5)$ is a Pythagorean triple, since $3^2+4^2 = 9+16 = 25 = 5^2$.

Exercise 4

Write a function `isTriple` that tests for Pythagorean triples. You don't need to worry about triples with sides of negative or zero length.

(a) Find the skeleton declaration of `isTriple :: Int -> Int -> Int -> Bool` and replace `undefined` with a suitable definition (use '`==`' to compare two values).

(b) Load the file into the REPL. Test your function on some suitable input numbers. Make sure that it returns `True` for numbers that satisfy the equation (such as `3`, `4` and `5`) and `False` for numbers that don't (such as `3`, `4` and `6`).

```
Main> isTriple 3 4 5
True
Main> isTriple 3 4 6
False
```

Next we'll create some triples automatically. One simple formula for finding Pythagorean triples is as follows: $(x^2 - y^2,\ 2yx,\ x^2 + y^2)$ is a Pythagorean triple for all positive integers x and y with $x > y$. The requirements that x and y are positive and that $x > y$ ensure that the sides of the triangle are positive; for this exercise, we will forget about these constraints.

Exercise 5

Write functions `leg1`, `leg2` and `hyp` that generate the components of Pythagorean triples using the above formulas.

(a) Using the formulas above, add suitable definitions of

```
leg1 :: Int -> Int -> Int
leg2 :: Int -> Int -> Int
hyp  :: Int -> Int -> Int
```

to your `tutorial1.hs` and reload the file.

(b) Test your functions on suitable input numbers. Verify that the generated triples are valid.

```
Main> leg1 5 4
9
Main> leg2 5 4
40
Main> hyp 5 4
41
Main> isTriple 9 40 41
True
```

QuickCheck

Now we will use QuickCheck to test whether our combination of `leg1`, `leg2`, and `hyp` does indeed create a Pythagorean triple. QuickCheck can try your function out on large amounts of random data, which it creates itself. It's always a good idea to thoroughly test your code, and a better idea to have an automatic way to do that! But before we start using QuickCheck, we will try to get a flavour of what it does by testing your functions manually.

Exercise 6

The function `prop_triple`—the name starts with `prop`(erty) to indicate that it is for use with QuickCheck—uses the functions `leg1`, `leg2`, `hyp` to generate a Pythagorean triple, and uses the function `isTriple` to check whether it is indeed a Pythagorean triple.

(a) How does this function work? What kind of input does it expect, and what kind of output does it generate?

(b) Test this function on at least 3 sets of suitable inputs. Think: what results do you expect for various inputs?

(c) Type the following at the REPL-prompt (mind the capital 'C'):

```
Main> quickCheck prop_triple
```

The previous command makes QuickCheck perform a hundred random tests with your test function. If it says:

```
OK, passed 100 tests.
```

then all is well. If, on the other hand, QuickCheck responds with an answer like this:

```
Falsifiable, after 0 tests:
5
6
```

then your function failed when QuickCheck tried to evaluate it with the values 5 and 6 as arguments—when testing manually, that would be:

```
Main> prop_triple 5 6
False
```

If this happens, at least one of your previous functions `isTriple`, `leg1`, `leg2` and `hyp` contains a mistake, which you should find and correct.