University of Canberra
Faculty of Information Sciences and Engineering

Computer Engineering 1 — 1997
Part 1 : Functional Programming

Practical Class 2 — Week 3

Aim

The aim of this class is to help you learn to define simple Haskell functions. Also, you will be introduced to some common error messages produced by Hugs.

Getting Started

In this week's class you will construct some of your own Haskell function definitions and also modify a script that I have provided. To keep your files organised, it is probably a good idea to have a separate subdirectory for each week's lab class. Copy the `labw3` folder from the `Scripts` directory to your home directory. (You learned how to do that last week.) You should use `labw3` as your working directory for this week's exercises.

Exercise 1 — Area of a Triangle

The area of a triangle with sides a, b, c is given by the formula:

$$\sqrt{s(s-a)(s-b)(s-c)}$$

where

$$s = \frac{a+b+c}{2}$$

Design a Haskell function `triangleArea` to calculate the area of a triangle given the lengths of its sides. The definition above should suggest the use of a *where* clause.

The type of `triangleArea` should be `Float -> Float -> Float -> Float`.

The behaviour of `triangleArea` should be as follows:

```
Hugs session for:
/pub/local/Hugs/lib/Prelude.hs
week3.lhs
? triangleArea 3 4 5
6.0
? triangleArea 1 2 2.5
0.949918
? triangleArea 1 1 (sqrt 2)
0.5
?
```

Exercise 2 — Sum Test
Design a Haskell function `isSum` that takes three integer arguments and tests whether one of them is the sum of the other two.

The behaviour of `isSum` should be as follows:

```
Hugs session for:
/pub/local/Hugs/lib/Prelude.hs
week3.lhs
? isSum 1 2 3
True
? isSum 4 9 5
True
? isSum 12 5 7
True
? isSum 23 23 23
False
?
```

What should the type of `isSum` be? (Remember that you are expected to declare the type of `isSum` in your script, along with the definition.)

Exercise 3 — Hugs Error Messages
The purpose of this exercise is to show you some common Hugs *error messages* and to get you to interpret them and correct them. The script `errors.lhs` provided in the `labw3` directory contains a few simple and common errors. You should begin by loading `errors.lhs` into Hugs. Hugs will respond by reporting an error and a line number where the error was found. Using the editor, you should find that line, try to figure out what the error is, and correct it. Then reload the script to find the next error. You can just type `:reload` or even just `:r` at the Hugs prompt to reload the script.

Note that getting used to the error messages and what they really mean is an important (and often quite difficult) part of learning to use a programming language effectively. Often the messages require deep knowledge of the language to understand them fully. In languages like Haskell, the type mismatch information can be hard to fathom. If you have found all the errors in `errors.lhs`, I expect that you are wondering why an indentation error is reported as: `Syntax error in declaration (unexpected ';')`. My best suggestion for now is that you build on your experiences to learn to *recognise* and *interpret* the error messages.

Exercise 4 — Fibonacci Numbers
Early in the 13th Century, Leonardo Fibonacci described the following sequence:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots$$

Except for the first two numbers in the sequence, each is the sum of the two numbers preceding it in the sequence.

Construct a Haskell function to calculate the nth element of the sequence of Fibonacci numbers. For example:

```
Hugs session for:
/pub/local/Hugs/lib/Prelude.hs
week3.lhs
? fibonacci 1
0
? fibonacci 2
1
? fibonacci 3
1
? fibonacci 8
13
? fibonacci 20
4181
?
```

You should take a similar approach to that used in the `factorial` function shown in lectures. To help jog your memory, here is the definition using guards:

```
>  factorial :: Int -> Int
>  factorial n
>    | n == 0      = 1
>    | otherwise   = n*(factorial (n-1))
```

`fibonacci` is more complicated than `factorial`, so here is a hint. Your function definition will need to handle three cases: $n = 1$, $n = 2$ and $n > 2$.

Make sure that your script contains appropriate comments as well as the definitions and type declarations of your functions.

When you have completed exercise 1–4 above, you should demonstrate them to your tutor and ask for their comments. Remember that attendance at Practical classes and satisfactory completion of the exercises is part of your assessment.

Extension

The following two exercises are fairly straightforward extensions of those above. While they are not compulsory, I recommend that you attempt them sometime in the next week or two. They will reinforce what you have learnt so far.

Extension Exercise 5 — Area of a Triangle (revisited)

Have you considered what your `triangleArea` function from exercise 1 will do with invalid data? For example, there is no triangle with sides $1, 2, 4$. What does Hugs give as the value of the expression: `triangleArea 1 2 4`?

Add to your function definition some checks to handle such invalid data and report an error if appropriate.

Hint: Look back at the `roots` (of a quadratic) function discussed in lectures for ideas about how to check valid data and use the `error` function.

Hint: Luckily, your `isSum` function from exercise 2 is pretty close to what you need for checking that three numbers can really be the sides of a triangle. Begin by modifying this function.

Extension Exercise 6 — Fibonacci (revisited)

It would be better form to define the `fibonacci` function using patterns rather than guards. Do so.