

## Week 3 Practical Class Exercises

### Defining Simple Functions in Haskell

---

#### Objectives

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

#### 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. For example, you could make a directory `labw03` as a subdirectory of the `comp1100` directory you made last week. Copy the files from the `labs/week03/` sub-directory of `/dept/dcs/comp1100/public/` to your this new directory. (You learned how to do that last week.) You should use `labw03` as your working directory for this week's exercises. The functions you are asked to define in the following exercises will be in a new script that you create. Don't forget that the name of the file must have a `.hs` suffix.

#### 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 the function should be `triangleArea :: Float -> Float -> Float -> Float`.

The behaviour of `triangleArea` should be as follows:

```
*Main> triangleArea 3 4 5
6.0
*Main> triangleArea 1 2 2.5
0.94991773
*Main> triangleArea 1 1 (sqrt 2)
0.5000001
```

### 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. 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.)

The behaviour of `isSum` should be as follows:

```
*Main> isSum 1 2 3
True
*Main> isSum 4 9 5
True
*Main> isSum 23 23 23
False
```

### Exercise 3 (GHCi error messages)

The purpose of this exercise is to show you some common GHCi *error messages* and to get you to interpret them and correct them. The script `errors.lhs` provided in the `lab/week03` directory contains a few simple and common errors. You should begin by loading `errors.lhs` into GHCi. GHCi will respond by reporting an error and a line number where the error was found. *Note that the error may in fact be on the line before, or in some cases even further away from the point reported by GHCi. The error message will not necessarily tell you what your mistake is. Rather, it will tell you why GHCi can't go any further.*

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 type `:reload` or even just `:r` at the GHCi prompt to reload the script. *If you see an error other than the one just reported by GHCi, don't correct it. The purpose of the exercise is to see the error messages.*

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 a deep knowledge of the language to understand them fully. In languages like Haskell, the type mismatch information can be hard to fathom. My best suggestion for now is that you build on your experiences to learn to *recognise* and *interpret* the error messages.

### Exercise 4 (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 GHCi give as the value of the expression: `triangleArea 1 2 4`? Add to your function definition, a check 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 be the sides of a triangle, but you'll have to think about it. Begin by modifying this function.

**Show your tutor your completed work.**

***Make sure you terminate your session by clicking the logout button on the front panel.***