CSC1048 Computability and Complexity

Functional Programming Lab 2: Defining Functions

Aim

The aim of this practical is to help you learn to define simple Haskell functions.

A function should have a type declaration and a body. Here is a simple function that returns the difference of two integers.

```
diff :: Int -> Int -> Int
diff x y = abs (x-y)
```

Why are the brackets important in the `diff` function?

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

$s = (a + b + c)/2$

Design a Haskell function `triangleArea` to calculate the area of a triangle given the lengths of its sides.

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

The behaviour of `triangleArea` should be as follows:

```
> triangleArea 3 4 5
6.0
> triangleArea 1 2 2.5
0.949918
> triangleArea 1 1 (sqrt 2)
0.5
```

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:

```
> isSum 1 2 3
True
> isSum 4 9 5
True
> isSum 12 5 7
True
> isSum 23 23 23
False
```

You should start by declaring the type of `isSum` in your script.

3. 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 some checks to handle such invalid data and report an error if appropriate. You can use the built-in `error` function for this purpose; it is called with a string argument which is the error message. For example:

```
error "Not a triangle!"
```

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