Defining simple functions

1.1 Consider the following Haskell function defintions.

```
incr :: Int -> Int
incr x = x+1

triple :: Int -> Int
triple x = 3*x

welcome :: String -> String
welcome name = "Hello, " ++ name ++ "!"

count :: String -> String
count str = show (length str) ++ " characters."
```

Experiment step-by-step calculation of the following expressions using the Haskelite and the GHCi interpreters.

(a) `incr (triple 3)`

(b) `triple (incr (3+1))`

(c) `triple (incr 3 + 1)`

(d) `triple (incr 3) + 1`

(e) `welcome "Harry" ++ welcome "Potter"`

(f) `welcome ("Harry" ++ " Potter")`

(g) `welcome (welcome "Potter")`

(h) `count "Expelliarmus!"`

(i) `count (count "Expelliarmus!")`

1.2 Using the Prelude functions `length`, `take`, `drop` presented in Lecture 1, define two functins `leftHalf` e `rightHalf` that divide a list into two approximate halves. Examples:

```
leftHalf [1,2,3,4,5,6,7] == [1,2,3]
rightHalf [1,2,3,4,5,6,7] == [4,5,6,7]
```

Try your definitions on the GHCi interpreter and investigate what happes if the list has a single element or is empty.

1.3 Use the Prelude functions presented in Lecture 1 (`head`, `tail`, `length`, `take`, `drop` and `reverse`) to answer the following questions.

(a) Define a function to get the second element from a list. Example:

```
second [1,2,3,4] == 2
```

Investigate what happens if the list has fewer than 2 elements.

(b) The function `last` gets the last element of a list. Example:

```
last [1,2,3,4] == 4
```

Show that this function can be defined as a composition of the above functions. Can you find two distinct definitions?

(c) Define the `init` function that removes the last element from a list using the above functions. Example:

```
init [1,2,3,4] == [1,2,3]
```

Can you find two distinct definitions?

(d) Define a `middle` function that gives que the middle element in a list. Example:

```
middle [3,2,1,4,5] == 1
```

Investigates what happens if the list has an even or odd number of elements.

(e) Define a function `checkPalindrome` that checks if a string is a *palindrome*, i.e. if it is equal to its reverse. The result should be a truth value (`Bool`). Examplos:

```
checkPalindrome "abba" == True
checkPalindrome "abra" == False
```

1.4 The following conditions should hold for three positive values to be sides of a triangle: any of the values must be smaller than the sum of the other two. Complete the following definition of a function that tests these conditions; the result should be a boolean value (`True` or `False`).

```
checkTriangle :: Float -> Float -> Float -> Bool
checkTriangle a b c = ...
```

Example: `checkTriangle 3 6 2 == False`

1.5 We can compute the area A of a triangle whose sides measure a, b, c using the following formula:

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

where $s = (a+b+c)/2$. Complete the following Haskell function definition to compute this area.

```
triangleArea :: Float -> Float -> Float -> Float
triangleArea a b c = ...
    where s = ...
```

Example: `triangleArea 3 4 5 == 6.0`

Types and classes

1.6 Match each of the expressions in the left hand side with its admissable type on the right hand sides.

(a) ('a','2')	(1) Bool
(b) ('b',1)	(2) [Char]
(c) ['a','b','c']	(3) (Char,Char)
(d) 1+2 == 4	(4) (Char,Int)
(e) not	(5) [(Bool,Bool)]
(f) sqrt	(6) ([Bool],[Bool])
(g) [sqrt, sin, cos]	(7) Float -> Float
(h) [tail, init, reverse]	(8) Bool -> Bool
(i) ([False,True],[True,False])	(9) [[a] -> [a]]
(j) [(False,True),(True,False)]	(10) [Float -> Float]

1.7 Mark all the following expressions that have a type error.

(a) 1 + 1.5

(b) 1 + False

(c) 'a' + 'b'

(d) 'a' ++ 'b'

(e) "a" ++ "b"

(f) "1+2" == "3"

(g) 1+2 == "3"

(h) show (1+2) == "3"

(i) 'a' < 'b'

(j) 'a' < "ab"

(k) (1 <= 2) <= 3

(l) (1 <= 2) < (3 <= 4)

(m) head [1,2]

(n) head (1,2)

(o) tail "abc"

1.8 Determine the most general type for each of the following definitions. You should include type class restrictions for any overloaded operations.

(a) second xs = head (tail xs)

(b) swap (x,y) = (y,x)

(c) pair x = (x,x)

(d) double x = 2*x

(e) half x = x/2

(f) average x y = (x+y)/2

(g) isLower x = x>='a' && x<='z'

(h) inRange x lo hi = x>=lo && x<= hi

(i) isPalindrome xs = xs == reverse xs

(j) twice f x = f (f x)