# Worksheet 4 Function application Answers

**Task 1**

1. The process of giving particular inputs to a function is known as:

 function application

2. Without using a computer, think about the function

 times x y z = x \* y \* z

 If we then define

fiveDeep l w = times 5 l w

 What is returned by the following?

 fiveDeep 3 4

 \*Main> fiveDeep 3 4

 60

 ***(See file fiveDeep.hs)***

3. Create and load a file with the functions **times** and **fiveDeep.** What happens when you enter **:type** for each of these at the prompt? Why?

 \*Main> :type times

 times :: Num a => a 🡪 a 🡪 a 🡪 a

 \*Main> :type fiveDeep

 fiveDeep :: Num a => a 🡪 a 🡪 a

 times takes three arguments and returns a value. a is a **type variable**, which means that a can be of any type. Functions that have type variables are called **polymorphic** functions.

 fiveDeep has one less argument, because the value 5 is bound to it.

### Task 2: map and filter

4. Write down what you expect to the returned by the following. Use a calculator if you need to. Then test in Haskell.

 (a) map (\*5) [4,9,15]

 (b) map sqrt [4,9,16,37]

 (c) map sin [0,pi/4, pi/2]

 \*Main> map (\*5) [4,9,15]

 [20,45,75]

 \*Main> map sqrt [4,9,16,37]

 [2.0,3.0,4.0,6.082762530298219]

 \*Main> map sin [0,pi/4, pi/2]

 [0.0,0.7071067811865475,1.0]

5. Create and save a file filter.hs to implement the function

 isEven n = n `mod` 2 == 0

 (Hint: Use the backward quotes on the left of the 1 key on the keyboard to surround the mod operator. )

-- filter.hs

isEven n = n `mod` 2 == 0

listA = [1,2,3,4,5,6]

 Use **map isEven** to test it on a list of half a dozen integers. What is output?

**\*Main>** map isEven listA

[False,True,False,True,False,True]

 What would you expect to happen if you tried an argument which is not an integer? Try it.

 Haskell will return an error.

\*Main> isEven "a"

<interactive>:123:1: error:

 • No instance for (Integral [Char]) arising from a use of ‘isEven’

 • In the expression: isEven "a"

 In an equation for ‘it’: it = isEven "a"

6. Explore the use of **filter (>"g")** with a list of characters.

 \*Main> filter (>"g")["A","B","a","b","h","n"]

 ["h","n"]

 \*Main>

 (“A…Z” and “ a…g” are all <= “g”. )

### Task 3

7. Use a fold operation to find the sum of the elements in [1,2,3,4,5]

\*Main> foldr (+) 0 [1,2,3,4,5]

15

(or use foldl which gives the same result)

8. Adapt your code from question 6 to find the factorial of 5, i.e. (5 x 4 x 3 x 2 x 1)

\*Main> foldr (\*) 1 [2,3,4,5]

120

(or use foldl which gives the same result; foldr (\*) 2 [3,4,5] also gives
the correct result.

A useful data type in Haskell is an ordered pair, called a **tuple**. Some examples are:

(1,2), (5.8, True) and (“Helen”, 15)

9. Define a tuple **t1** from any of these examples. Try out the built-in functions **fst** and **snd** on t1. The functions return the first and second elements respectively

\*Main> let t1 = ("Helen", 15)

\*Main> fst t1

"Helen"

\*Main> snd t1

15

\*Main>

10. Define your own list of tuples, with a consistent type or pair of types. Write down the list, and the result of applying **map fst** or **map snd** to your list.

 Examples:

\*Main> let list1 = [(1,2), (3,4), (5,6)]

\*Main> map fst list1

[1,3,5]

\*Main> map snd list1

[2,4,6]

**\*Main>** let list2 = [(1,"Bob"),(2,"Jo"),(5,"Mary")]

**\*Main>** map fst list2

[1,2,5]

**\*Main>** map snd list2

["Bob","Jo","Mary"]

**\*Main>**

11. Define a list of tuples of pairs of numbers. Use a **fold** function to calculate the sums of the first elements from every tuple.

Example: foldl (+) 0 (map fst list1)

12 Write functional programming code in Haskell or another language, including type declarations, which takes a list of the first 100 integers and returns the odd numbers which are multiples of 3.

 Hint: the definition **listN = [1..N]** defines a list of all the integers between 1 and N.

 Example

isOdd::Integer 🡪 Bool

isOdd x = not (x `mod` 2 == 0)

isMultof3::Integer 🡪 Bool

isMultof3 x = x `mod` 3 == 0

list100 = [1..100]

filter (isOdd) (filter (isMultof3) list100)

Expected return: [3,9,15,21,27,33,39,45,51,57,63,69,75,81,87,93,99]

 ***(See file oddThrees.hs)***