COMP105: Programming Paradigms
Week 5 Homework Sheet

This is the homework sheet for **Week 5**. Complete your answers in a file named `week5.hs` and submit them to the "Week 5" assessment in SAM here

`https://sam.csc.liv.ac.uk/COMP/Submissions.pl`

Submission of the weekly homework sheets contributes 10% of the overall module mark, and each homework sheet counts equally towards this. Each homework sheet will be marked on a pass/fail basis. You will receive full marks for submitting a *reasonable attempt* at the homework. If no submission is made, or if a non-reasonable attempt is submitted, then no marks will be awarded.

The deadline for submission is

Friday Week 5 (13/11/2020) at 16:00.

Late submission is **not** possible. Individual feedback will not be given, but full solutions will be posted promptly after the deadline has passed.

If you feel that you are struggling with the homework, or if you have any other questions, then you can contact the lecturer at any point during the week via email, or you can drop in to the weekly Q&A session on MS Teams on Friday between 1PM and 4PM.

Lecture 12 - Partial application. All of the questions below ask you to use partial application to create a new function. For example, if you were asked to use partial application to write a function `f` that added one to its input, the answer would be

```
f = (+1)
```

Note that here we do not specify any arguments for `f`, instead we build `f` by partially applying `+`.

1. Use partial application with the `+` function to write a function `plus_ten` that returns its input plus 10.

2. Use partial application with the `==` function to write a function `is_twenty` that returns `True` if its input is 20, and `False` otherwise.

3. Use partial application with the `**` function to write a function `three_power` that takes one argument `n` and returns 3^n.

4. Use partial application with the `**` function to write a function `power_three` that takes one argument `n` and returns n^3.

5. Copy the code below into your file:

   ```
   xisy x y = x ++ " is " ++ y
   ```

   Partially apply `xisy` to write a function `cakeis` that takes one argument `x` and returns `"cake is " ++ x`

Lecture 13 - Polymorphic Types. In a comment in your file, write down the most general type annotations for each of the following functions. Remember that if you get stuck, you can use the `:type` command to ask ghci what it thinks the type should be. You can also use this to check your answers. Make sure that you understand why ghci has assigned that type to the function.

1. `func1 a b = a + b + 2`
2. `func2 a b = (a `div` 2, b / 2)`
3. `func3 (x:y:xs) = x == y`
4. ```
   func4 [] = []
   func4 (x:xs)
       | x > 0     = x : func4 xs
       | otherwise =     func4 xs
   ```

Lecture 14 - Anonymous functions. All of the questions below ask you write an anonymous function. You should do this in ghci, and then test the functions by giving the arguments like so:

```
ghci> (\ x -> x + 1) 10
11
```

Write your anonymous functions in a comment in your file.

1. Write an anonymous function that takes a number `x` and returns `x-1`.
2. Write an anonymous function that takes two arguments `x` and `y` and returns `show x ++ show y`.
3. Write an anonymous function that takes a two-element tuple `(x, y)` and returns `(y, x)`.
4. Write an anonymous function that takes a list, and returns the second element of the list.

Lecture 14 - Function composition. Use the `.` and `$` operators to re-write the following ghci queries. Put your modified queries in a comment in your file.

1. `head (head [[1]])`
2. `(+1) ((*2) 4)`
3. `sum (tail (tail [1,2,3,4]))`
4. `filter (>10) (map (*2) [1..10])`
```

Lecture 15 - Map.

1. Use `map` to write a function `triple list` that takes a list of numbers, and returns a list where each number is multiplied by three.
2. Use `map` to write a function `list_to_str` that takes a list of numbers, and returns a list where each number has been converted to a string.
3. Use `map` to write a function `second_char` that takes a list of strings, and returns a list containing the second character of each of the input strings.
4. Use `map` to write a function `add_pairs` that takes a list of pairs of numbers, and returns a list that contains the sum of each pair.

Lecture 15 - Filter.

1. Use `filter` to write a function `only_odds` that takes a list of numbers, and returns a list containing only the odd numbers in the list.
2. Use `filter` to write a function `between a b list` that takes two numbers `a < b` and a list of numbers, and returns only the numbers that are strictly between `a` and `b`.
3. Use `filter` to write a function `ordered` that takes a list of pairs, and returns only the pairs `(a, b)` for which `a > b`.
4. Use `filter` to write a function `singletons` that takes a list of lists, and returns only the lists that have exactly one element.