COMP105: Programming Paradigms
Lab Sheet 5

This lab covers partial application, anonymous functions, map, and filter.

1. **Check the hidden tests from last week.** If you've not done so already, check the results of the hidden tests from last week's challenge lab. Did you fail any of these tests? Some of the tests check edge cases that are legal inputs that you might not have expected.

2. **Partial application.**

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

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

   (c) Use partial application with the `**` function to write a function `three_power` that takes one argument `n` and returns 3^n. **Make sure that you use `**` and not `^`, or your code will not compile on Codegrade.**

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

   (e) Use partial application with the `take` function to write a function `take_four` that takes a list and returns the first four elements of that list.

3. **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
```

   (a) Write an anonymous function that takes a number `x` and returns `x-1`.

   (b) Write an anonymous function that takes two arguments `x` and `y` and returns `x+y`.

   (c) Write an anonymous function that takes two arguments `x` and `y` and returns `show x ++ show y`.

   (d) Write an anonymous function that takes a two-element tuple `(x, y)` and returns `(y, x)`.

   (e) Write an anonymous function that takes a list, and returns the second element of the list.

4. **Map.**

   (a) 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.

   (b) 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.

   (c) 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.

   (d) 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.

   (e) Use a nested `map` to write a function `triple_list_list` that takes a list-of-lists containing numbers, and triples each number.

5. **Filter**.

   (a) 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.

   (b) Use `filter` to write a function `vowels` that takes a string, and returns only the vowels (`"aeiou"`) in the string.

   (c) 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`.

   (d) Use `filter` to write a function `ordered` that takes a list of pairs, and returns only the pairs `(a, b)` for which `a > b`.

   (e) Use `filter` to write a function `singletons` that takes a list of lists, and returns only the lists that have exactly one element.

   (f) Use `map` and `filter` to write a function `only_odds_list` that takes a list of lists of numbers, and returns a list of lists of numbers containing only the odds numbers from the input.

6. **Function composition.** Use the `.` operator to re-write the following ghci queries.

   (a) `head (head [[1]])`

   (b) `(+1) ((*2) 4)`

   (c) `sum (tail (tail [1,2,3,4]))`

   (d) `filter (>10) (map (*2) [1..10])`

7. **Function types.** Annotate each function in Questions 4 and 5 with a type. Load the code into ghci and check that it compiles. Comment out your type annotation and then ask ghci for the type of the function. Does ghci give it a more general type? Note: ghci may use type-classes that we have not covered in COMP105.

Lab complete.