# COMP105: Programming Paradigms
# Week 10 Homework Sheet

This is the homework sheet for **Week 10**. Complete your answers in a file named `week10.hs` and submit them to the "Week 10" 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 10 (18/12/2020) 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 26 - Lazy evaluation** Consider the following function.

```
f b x y z = if b then div x y else z
```

Which of the following inputs would return an error? You can check using ghci whether you are correct. Write your answer down in your file in a comment, and give a short justification for your answer.

1. `f True 1 0 3`
2. `f True 1 2 (error "error")`
3. `f False 1 0 3`
4. `f False 1 0 (error "error")`
5. `f True 1 2 (f False 1 2 (error "error"))`

**Lecture 26 - Lazy evaluation on lists** Which of the following queries terminates? You can check your answers by typing the queries into ghci (remember that control + c will terminate an infinite computation). Give a short justification for your answer.

1. `take 4 [1..]`
2. `drop 4 [1..]`
3. `map (*2) [1..]`
4. `(zipWith (+) [1,3..] [2,4..]) !! 1000`

**Lecture 27 - Tail recursion** For each of the following questions, write an *efficient* function that uses tail recursion.

1. Write a tail recursive function `product' :: [Int] -> Int]` that takes a list of numbers and multiplies all of the numbers together.
2. Write a tail recursive implementation of `sum_up_to :: Int -> Int` that takes a number n, and adds up all of the numbers between 0 and n inclusive.
3. (*) Write a tail recursive implementation of `even_sum :: Int -> Int` that takes a number n, and adds up all of the even numbers less than or equal to n.

**Lecture 27 - Folds** For each of the following, select which of `foldr`, `foldl`, `foldl'` would be the *best* choice to implement the function in Haskell. Write your answer in your file, and give a short justification for your choice. (There is no need to code the function yourself).

There are two points to bear in mind here. Firstly, will the function consume every element of the list? If so, strict evaluation should be preferred. Secondly, could the function possibly be called on an infinite list and still produce an answer? If so, preserving laziness should be preferred.

1. A function `even_product :: [Int] -> Int` that takes a list of integers and multiplies all of the even elements of that list together.
2. A function `sum_fsts :: [(Int, String)] -> Int` that takes a list of pairs, and returns the sum of the first elements of the pairs.
3. A function `even_elements :: [Int] -> [Int]` that takes a list of integers and returns a list containing all of the integers that are even (filter might be a better choice, but you could do this with a fold).