# COMP105: Programming Paradigms
# Week 3 Homework Sheet

This is the homework sheet for **Week 3**. Complete your answers in a  le named `week3.hs` and submit them to the \Week 3" 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 3 (30/10/2020) 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.

**Week 3.** This week we will be learning about recursion during all three lectures, so all of the questions here are on that topic. Unlike the previous two weeks, the questions for Lecture 8 are generally harder than the questions for Lecture 7, and likewise the questions for Lecture 9 are harder than those for Lecture 8.

As always, you do not have to solve everything, and you just need to submit a reasonable attempt (having spent 1{3 hours on the questions). In short, if you are still struggling on Lecture 7 or Lecture 8 questions after having spent a couple of hours, just submit what you have, and perhaps include some comments about what you are stuck on. That would very likely meet the reasonable attempt requirement.

## Lecture 7 { Recursion.

1. Write a recursive function `mult13 n` that computes $13 * n$ by adding 13 to itself $n$ times.

2. Write a recursive function `pow3 n` that computes $3^n$ by multiplying 3 by itself $n$ times.

3. Write a recursive function `odd_sum n` that computes the sum of all odd numbers less than or equal to `n`. Hint: modify the `even_sum` function from Lecture 7 if you are stuck.

4. The *Lucas numbers* (denoted by $L_i$) are de ned so that $L_0 = 2$, $L_1 = 1$, and $L_n = L_{n-1} + L_{n-2}$ for all $n > 1$. Write a simple recursive function `lucas n` that computes the $n$th Lucas number. Your solution does not have to be computationally e cient. (Hint: try modifying the Fibonacci function that we saw in the lecture).

## Lecture 8 { List Recursion

1. Write a recursive function `half_sum list` that computes the sum of all elements in the list divided by 2.

2. Write a recursive function `mult2 list` that multiplies all elements of the input list by 2.

3. Write a recursive function `drop_evens list` that returns a new list with only the odd elements from the input list.

4. Write a recursive function `mult_adjacent list` that takes a list with an even number of elements, and returns a new list where adjacent pairs have been multiplied together. So, `mult_adjacent [1,2,3,4]` should return `[2, 12]`. (Hint: try modifying the `add_adjacent` function from the lecture).

5. Write a recursive function `get_ele i list` that returns the element at position `i` in `list`. Your function should return an error if the list does not contain `i` elements.

6. Write a recursive function `drop_ele i list` that returns a copy of the input list with the element at position `i` removed. Your function should return an error if the list does not contain `i` elements.

## Lecture 9 - More complex recursion on lists.

1. Write a recursive function `div_list list1 list2` that takes two lists of the same length, and divides each element in `list1` by the corresponding element in `list2`. So the function returns a list where the element in position `i` in the output is `(list1 !! i) / (list2 !! i)`.

2. Write a recursive recursive function `longer list1 list2` that returns `True` if `list1` is longer than `list2`, and `False` otherwise.

3. Write a recursive function `vowels_and_consonants string` that takes a string of characters and returns a pair of strings: the rst string in the pair should contain all vowels in the string (which are the letters `"aeiou"`), and the second string in the pair should contain all letters that are not vowels. You should use the `elem` library function in the solution for this question.

**Challenge question { The Collatz problem.** This is a very di cult question. Only attempt it if you really want to push yourself.

Suppose we take a number $n$ and do the following transformation:

- $n \to n/2$ if $n$ is even,
- $n \to 3n + 1$ if $n$ is odd.

Applying this rule starting with $n = 13$ and stopping when we reach 1 generates the following sequence

$$13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1.$$

We call this the *Collatz sequence* for 13, and it has 10 terms (including 13 and 1 themselves). Starting at di erent numbers gives Collatz sequences of di erent lengths. For example, the sequence starting at $n = 97$ has 119 terms, which is the longest sequence with starting number less than 100.

It is thought that every Collatz sequence will eventually arrive at 1 (the Collatz conjecture,) but it has not yet been proved!

Write a function `longest_collatz n` that computes the starting number less than `n` that has the longest Collatz sequence. If there is a tie, then return the largest number that is tied for the longest sequence length. Your solution will probably need to use one or more helper functions. The `div` function does integer division.