1. Write a four-stage function composition in Erlang, such that:
   a. The composition uses a list comprehension that outputs a stream of integers from 1 to M (where M is a parameter) to the first stage in the composition.
   b. The first stage increments integers from the first stage by 1.
   c. The second stage squares the integers from the second stage.
   d. The third stage sums up all the integers received.

2. Write a four-stage parallel pipeline using Skel, such that:
   a. The input to the pipeline is a stream of integers from 1 to M (where M is a command line parameter).
   b. The first stage increments integers from the first stage by 1.
   c. The second stage squares the integers from the second stage.
   d. The final stage sums up all the integers received.

3. Write the same program as (2) but using Erlang spawn and receive directory. Use the syntax of `spawn` and `receive` from the first lecture.

4. Assume the computations of the increment and square operations (from the previous exercise) take 0.2 seconds each. Use the `fib:fib` function to act as a payload. Experiment with the parameter to fib until you get aprox., 0.2 seconds. Use `timer:tc` to profile your Erlang program.

5. Assume increment takes 0.1 seconds, while square takes 0.5 seconds
   e. Modify the program from 2 until you get this desired effect (modify the parameter to `fib`)
   f. Further transform/tune the previous parallelization until you get an optimal performance by using skel.

6. Download http://chrisb.host.cs.st-andrews.ac.uk/ant_colony.zip
   g. Run the code and observe the performance
   h. Parallelise the code using Skel. HINT. You will need to extract functionality into functions and use a Pipeline, with nested skeletons in the stages.