Introduction to Scala

Practical Exercises

Some Functional Programming Techniques

1. Implement a function called **`timed`**, which allows a caller to measure the time taken to evaluate an expression passed to it as an argument. In other words:

   ```
   val timeToExecute = timed { doSomething }
   ```

   For the moment, assume the expression (**`doSomething`**) is of type **`Unit`**.

   Use the function System.nanoTime to find the time in nanoseconds. This works like System.currentTimeMillis, but at a higher resolution. It may not always be accurate to the level of nanoseconds (this is platform dependent) however it is useful for measuring time intervals.

2. The Fibonacci series can be defined as follows:

   $X_0 = 0$
   $X_1 = 1$
   For $n > 1$, $X_n = X_{n-2} + X_{n-1}$

   We can implement the Fibonacci series in a variety of different ways. For example, the definition implies a recursive approach. Write a (tail) recursive function that returns the n^{th} Fibonacci number.

   Streams appear to offer a more attractive solution.Define a Stream of Int values that contains the Fibonacci series. For a given n, how would you use this Stream to return the n^{th} Fibonacci number?

   Use your function timed to measure the relative performance of the two solutions.

3. Using the technique shown in the slides, implement the **`while`** loop as a curried function. How would you change this to provide an "**`until`**" loop – like the **`while`** loop but it executes until the condition evaluates to **`true`**?

   Can you provide the functionality of the **`do`**…**`while`** loop in this way? This also behaves like **`while`**, but the test is performed *after* each iteration of the loop.