**TCSS 380, Autumn 2022, Lab 1**

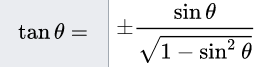
The objective of this lab assignment is to illustrate in Erlang some of the textbook concepts that appear in chapters 6, 9, and 15: basic syntax of a functional language.

You should work with a partner to complete this lab according to the instructions provided via Canvas. You should have all the required software installed and tested before starting this lab. Each file that you submit is to have both your and your partner’s names listed at the top of it. Submit only once per partnership. If you complete the lab or a part of it on your own, list your name only and provide a brief explanation as to why you were unable to work with a partner. Altogether, you need to upload into Canvas:

• lab01.erl  
• screenshot/s of running lab01.erl in the shell showing the function calls (png, jpg, or gif)

In the text editor of your choosing create a file **lab01.erl** and write function definitions that could be run from the shell – make sure you use the exact same names (anything in italics) as indicated in function descriptions. Pay attention to directions that specify what syntactical mechanisms to use within the functions (the purpose of this lab is to make sure you are familiar with most of the basic Erlang syntax discussed this week).

1. Function *myTan* that takes one argument that represents since of an angle and returns the tangent of that angle using the following equation:



* assume that in our case the denominator will be positive
* example: if the function is called with a value 0.57, then ~0.64 is retuned
* you do NOT have to guard against invalid input
* you may use: square root, sin, and pow functions defined in the math library but NOT tan() <https://erlang.org/doc/man/math.html>

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *isSame* that takes two arguments and returns **true** if these two values are the same considering both the value and the type and **false** otherwise. For example, if 5.0 and 5 are passed, the function should return false because these two parameters have different data types; if 5 and 5 are passed, the function should return true because these two parameters have the same value and data type.

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *parity* that takes an argument and returns an **atom** *even* if the argument is an even number, an **atom** odd if the argument is an odd number. Zero is even. Use function level pattern matching (several function clauses / headers with guards) instead of an *if* expression. Assume an integer will be passed, i.e. there is no need to handle non-integer values.

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *mealType* that takes an integer argument and returns its equivalent meal as an **atom**. Use function level pattern matching (several function clauses / headers) instead of a case expression inside a function. Use the following values:
   * for value 1 returns *breakfast*
   * for value 2 returns *lunch*
   * for value 3 returns *dinner*
   * for value 4 returns *appetizer*
   * for value 5 returns *dessert*
   * for an invalid value / type, returns *no\_match*

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *myMult* that implements the recursive math formula provided below – do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching

|  |  |  |  |
| --- | --- | --- | --- |
| myMult (m, n)= | page2image23339648 | (-1) \* myMult (m, n \* -1) | if n < 0 |
| 0 | if m == 0 or n == 0 |
| m + myMult(m, n-1) | if n > 0 |

When testing this function, *myMult(4, 3)* should result in 12, *myMult(4, -3)* should result in -12, *myMult(-4, 3)* should result in -12, and *myMult(-4, -3)* should result in 12.

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *mySum* that implements the following Java code as recursion in Erlang:

**int sum ( int boundary ) {**

**int sum = 0;**

**for(int i = 1; i <= boundary; i++)**

**sum += i;**

**return sum;**

**}**

Do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching. You do NOT need to guard against invalid input

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *mySeries* that takes 3 arguments: the value of the first term of the geometric sequence, the common ratio (multiplier), and the term number for which the function is to compute and return a value. For example, if *mySeries* is called with 2, 3, 4, then it means that the geometric sequence is: 2, 6, 18, 54, ... and 54 should be returned. You do NOT need to guard against invalid input. Do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching

✓ Run the function in the Erlang shell and take and save a screenshot

**EOF**