TCSS 380, Autumn 2019, Lab 4

The objective of this lab assignment is to give you more practice with the basic Erlang syntax and environment

Work with a partner of your own choosing to complete this lab. The instructor will attempt to check off the first 6 functions and their runs during the lab and if you get checked off, you don't need to submit the code for these 6 functions (lab04a.erl, lab04b.erl).
At the end of the lab submit all the parts you complete during the lab session that haven't been checked off – one submission per partnership.
If you do not complete some part within the two-hour session, finish at home by the designated due date. For this part, make sure that there is a comment at the top of your file that specifies whether you worked together or alone, and there is a name of your partner, if any. Include screenshots of running the functions in the shell. All together this lab involves:

- screenshot/s of running lab04c.erl in the shell showing various function calls (png, jpg, or gif)
- lab04a.erl
- lab04b.erl
- lab04c.erl

In the text editor of your choosing create a file **lab04a.erl** and write function definitions that could be run from the shell – make sure you use the exact same names as indicated in function descriptions:

1. function *cone* that takes two arguments that represent radius and height of a cone (in this order) and returns volume of that cone – feel free to look up the formula (but not Erlang code) online; you do NOT have to guard against invalid input; use pi defined in the math library
2. function *isDifferent* that takes two arguments and returns false if these two values are the same considering both the value and the type, i.e. if 5.0 and 5 are passed, the function should return true because these two parameters have different data types; if 5 and 5 are passed, the function should return false because these two parameters have the same value and data type
3. function *larger* that takes two arguments and returns the larger one – use the *if* statement inside the function body; if elements have the same value, returns either one
4. function *isEven* that takes an argument and returns true if the argument represents an even number, false otherwise
    - Show the first four functions and their tests to instructor

In the text editor of your choosing create a file **lab04b.erl** and write function definitions that could be run from the shell – make sure you use the exact same names as indicated in function descriptions:

5. function *displaySigns* that takes two arguments and echo prints them to console with a message:
    - *both positive* if both arguments > 0,
    - *both nonpositive* if both arguments <= 0
    - *different signs* if one of the arguments > 0 and the other <= 0

    Use the format specifier that will be appropriate for either floats or ints
6. function *rating* that takes an integer value (0-4) and returns the rating as an atom; where 4 is equivalent to *excellent*, 3 *good*, 2 *fair*, 1 *poor*, 0 *unacceptable* - use a case expression inside the function body; if someone enters an invalid value, e.g. 14, your function should return an atom *no_such_rating*
    - Show the next two functions and their tests to instructor

In the text editor of your choosing create a file **lab04c.erl** and write function definitions that could be run from the shell – make sure you use the exact same names as indicated in function descriptions:

7. function *fibonacci* 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 (i.e. write the selection the way *factorial* is written in *demo.erl*)

$$\text{fibonacci (n)} \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{fibonacci(n-1) + fibonacci(n-2)} & \text{if } n > 1 \end{cases}$$

When testing this function, *fibonacci (6)* should result in 8, *fibonacci(10)* should result in 55, and. If n are < 0, return an atom *wrong_input*

8. function *mysum* that implements the following C code as recursion in Erlang:

```
int sum ( int boundary ) {
    int i, sum = 0;
    for(i = 1; i <= boundary; i++)
            sum += i;
    return sum;
}
```

You do NOT need to guard against invalid input

9. 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

- Upload lab04c.erl to Canvas and a screenshot of running tests of the last 3 functions in the shell

--- CONGRATS – YOU ARE DONE – UNTIL NEXT LAB THAT IS ☺ ---