TCSS 380, Winter 2020, Lab 4

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

You must work with a partner during the lab. Find a partner of your own choosing – this time it could be a person with whom you worked before. If you cannot find a partner, ask the lab instructor for help. Before the end of the lab submit all the parts you complete during the lab session – one submission per partnership. Remember to list both names at the top of each file on which you work together. At the very least half of the lab must be completed during the lab session to receive full credit. If you do not complete some part within the two-hour session, finish at home on your own or with your lab partner by the designated due date. If you complete this part of the lab on your own, list your name only. If you complete this part with your lab partner, list both names and submit only once per partnership. All together, you need to upload:

- lab04.erl
- screenshot/s of running lab04.erl in the shell showing various function calls (png, jpg, or gif)

In the text editor of your choosing create a file **lab04.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 constructs discussed this week). As for the function calls, it is enough that you capture about 4 function calls total of different functions (i.e. not 4 calls of the same function).

1. function *pendulum* that takes two arguments that represent the length of the pendulum and the local strength of gravity (in this order) and returns the period of that pendulum (the time taken for a complete cycle) using the following formula:

   $$T \approx 2\pi\sqrt{\frac{L}{g}}$$

   where L is length and g is gravity; 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 *largerLog* that takes two arguments, calculates their logs using math function *log* and returns the larger log – use the *if* statement inside the function body; if parameters result in the same log value, the function returns either one
4. function *display* that takes two arguments and echo prints them to console with a message:
   - *both even* if both parameters are even numbers,
   - *both odd* if both parameters are odd numbers,
   - *different* if one of the parameters is odd and the other one is even

   To write a boolean expression inside an if, use parenthesis around expressions, e.g. (X > 0) and (Y > 0). You may also solve this using nested ifs but then remember not to end a nested if with a period since the period indicates the end of a function.
   Use the format specifier that will be appropriate for either floats or ints.
5. function *primaryColor* that takes an integer value (1-3) and returns its equivalent color as an atom, where 1 is equivalent to *red*, 2 *green*, 3 *blue* - use a case expression inside the function body; if someone enters an invalid value, e.g. 14, your function should return an atom *no_match*

6. function *primaryColor2* that works in the exact same fashion as *primaryColor*, except uses function level pattern matching (several function clauses / headers) instead of a case expression inside a function.
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

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

   When testing this function, *fibonacci (6)* should result in 8, *fibonacci(10)* should result in 55, and. If n < 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

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