TCSS 380, Autumn 2019, Lab 5

The objective of this lab assignment is to give you more practice with Erlang tuples and list syntax and list logic in functional programming

Work with a partner of your own choosing to complete this lab but your partner must be a different person than during the last lab. The instructor will attempt to check off the first 4 functions and their runs during the lab. Overall, first 6 functions must be completed during the lab for full credit. At the end of the lab submit all the parts you complete during the lab session – 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 test file in the shell for out-of-the-class portion. All together this lab involves:

- lab05.erl (to be turned in at the end of the lab and again by the deadline)
- lab05_tests.erl
- screenshot of running lab05_test.erl in the shell outside of the lab session

In the text editor of your choosing create a file **lab05.erl** and write function definitions described below – make sure you use the exact same names as indicated in function descriptions. Download the file **lab05_tests.erl**, add Erlang unit test library to it, and as you work on your solutions, write unit tests for your functions (unit tests for the first three functions are provided).

1. function *translate* that that takes two tuples, where the first tuple represents x, y, and z coordinates of a point and the second tuple represent by how many units the point should be moved along x-axis, y-axis, and z-axis. The function calculates and returns the new point. For example if {1, 2, 3} and {0, -1, 4} are passed to this functions, it should return {0, 1, 7} (hint: to pattern match a function on a tuple consisting of 3 elements, you can use *functionName ({X, Y, Z}) ->* … type of a syntax)
2. function *isYounger* that takes two dates (each date is a 3-int tuple *dd, mm, yyyy*) and evaluates to true or false. It evaluates to true if the first argument is a date that comes after the second argument (e.g. 3/4/2015 comes after 3/30/2012, hence a person born in 2015 is younger). (If two dates are the same, the result is false.). Think about how you can use relational operators on entire tuples to simplify processing instead of writing nested ifs
3. function *incrementAll* that takes a list of numbers and returns the original list with all elements incremented by 1, e.g. if the original list contains [1, 2, 3], then [2, 3, 4] is returned; you are NOT allowed to use ++ or built-in functions or list comprehension (you are to use the cons operator)
4. function *mymax* that takes a list and returns its max value – use tail recursion to find list max and return an atom *empty_list* if a list is empty; you are NOT allowed to use built-in functions; add appropriate test cases to the test file that use *assertEqual* macro
    - Show the first four functions and the run of the test file to instructor
5. function *removeDiv3* that takes a list of integers and removes all values divisible by 3, e.g. if the original list contains [0, 3, 1, 2, 6], then [0, 1, 2] is returned; you are NOT allowed to use built-in functions or list comprehension; add appropriate test cases to the test file that use *true* matching pattern
6. function *generate* – takes three ints as arguments and generates a list of integers from arg1 to arg2 (inclusive) in increments indicated by arg3, e .g. if arg1 = 3, arg2 = 8, and arg 3 = 2, then the function returns [3, 5, 7]. If arg1 > arg2, returns an empty list; assume arg3 will be a valid number; you are NOT allowed to use built-in functions or list comprehension; add appropriate test cases to the test file
    - Show the next two functions and the run of your test file to instructor

7. function *calculateBill* that takes a list of tuples, where a tuple is of the form *name, price, tax_amount*, and returns a list of tuples, where each tuple is of the form *name, total, tip,*
   where *tip = price * 0.2* and *total = price + tax_amount + tip*
   add appropriate test cases to the test file
8. function *getnth* that takes a list and an int n and returns the nth element of the list, where the head of the list is the 1st element. You are only allowed to use list functions hd and tl in your solution – no other built-in functions are allowed. If the list is empty or n is invalid, return a tuple:
   {error, no_such_element}
   add the following test cases to the test file:
   getnth_1_test() -> true = {error, no_such_element} == lab05:getnth([ ], 2).
   getnth_2_test() -> true = {error, no_such_element} == lab05:getnth(["hello", "there"], 3).
   getnth_3_test() -> true = {error, no_such_element} == lab05:getnth(["hello", "there"], 0).
   getnth_4_test() -> true = "there" == lab05:getnth(["hello", "there"], 2).
   getnth_5_test() -> true = "there" == lab05:getnth(["hello", "there", "where"], 2).
   getnth_6_test() -> true = "where" == lab05:getnth(["hello", "there", "where"], 3).
   getnth_7_test() -> true = "where" == lab05:getnth(["hello", "there", "where", "here"], 3).
9. function *repeat* that takes a list of integers and a list of nonnegative integers and returns a list that repeats the integers in the first list according to the numbers indicated by the second list; add the following test cases to the test file – these test files also illustrate the logic of repetition to be followed
   repeat_1_test() -> true = [1, 2, 2, 2, 2, 2, 3, 3] == lab05:repeat([1, 2, 3], [0, 4, 1]).
   repeat_2_test() -> true = [ ] == lab05:repeat([ ], [0, 4, 1]).
   repeat_3_test() -> true = [1,2,3] == lab05:repeat([1, 2, 3], [ ]).
   repeat_4_test() -> true = [4,4,4,5,6] == lab05:repeat([4,5,6], [2]).

- Upload lab05.erl to Canvas
- Upload lab05_tests.erl to Canvas

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