Programming Paradigms

Assignment #10: Clojure

Due Tuesday 19th April, 11.59pm

Worth 10 points

This homework involves exercises from class plus some additional problems to solve.

Turn in each answer by cutting and pasting the code from your REPL screen into a single document or copying your .clj code. Show all steps.

1. Generate a sequence of numbers from 0 to 100 and return all the numbers between 10 and 20.

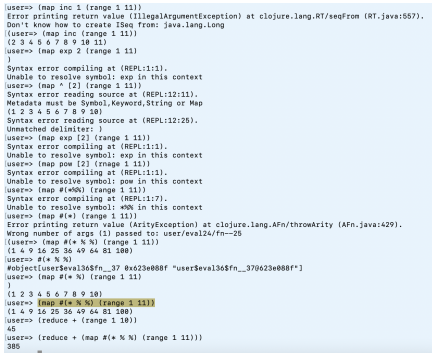
```
[user=> (take 11 (drop 10 (range 0 100)))
 (10 11 12 13 14 15 16 17 18 19 20)
```

2. Use range and map to generate a sequence of square numbers for 1 to 10 (inclusive)
   Display the sequence

```
[user=> (map #(* % %) (range 1 11))
 (1 4 9 16 25 36 49 64 81 100)
```

   Sum those results using reduce.

```
[user=> (reduce + (map #(* % %) (range 1 11)))
 385
```



3. Create a namespace called 'library' that includes a"tax" function. The function takes two values (1) the amount to be taxed and (2) the tax rate as a percent e.g., 10 and computes the tax.

   Use your new function in a println statement to report the tax that would be computed on an item that costs $117 with 7% tax.

   Call your function from a different name space called 'application'.

```
[mleonar8@student04:~$ vim question3.clj
[mleonar8@student04:~$ clojure -M question3.clj
 Tax on 117 at 17 is %n 19.89
[mleonar8@student04:~$ vim question3.clj
[mleonar8@student04:~$ more question3.clj
 (ns library)
 (defn tax [amountToBeTaxed, taxRate]
   (float (* (/ taxRate 100) amountToBeTaxed)))

 (ns application)
 (println "Tax on 117 at 17 is %n" (library/tax 117 17))
[mleonar8@student04:~$ clojure -M question3.clj
 Tax on 117 at 17 is %n 19.89
```

4. Write a function that generates numbers in the sequence
   3, 2, 5, 7, 12, ...

Note, you will have to figure out the sequence pattern so that you can determine what numbers come next and how to figure out the Nth number on the sequence.

Write assert (i.e., "is") tests for calling the function for the 4th, 6th, and 10th element. You must use the loop:recur construct. Hint: For the 10th element you may have to deal with an arithmetic overflow error. Remember you can change the type of a number e.g., 5 could be specified as 5N to make it a BigInt.

5. Run a performance test to compare the use of transient datastructures versus non-transient ones. Create a vector that contains all of the Lucas numbers from 0 to N. What are Lucas numbers? https://tinyurl.com/ya5h5utt How do you produce them? https://www.geeksforgeeks.org/lucas-numbers/

   Write a transient version and a non-transient version. Run performance tests at various values N. Remember the online IDE isn't always reliable for performance, so run a few times to "eyeball" a typical runtime. Examples were provided in class for timing.

6. Write a function that accepts a word, searches for the first occurrence of a 'double letter', and returns that letter. If no letter appears twice consecutively, return nil.
   E.g., "adds" returns 'd'. "Primrose" returns 'nil'

7. Write a one paragraph reflection on Clojure as a language. What do you think about it? How does it compare to other languages we've looked at.

   Clojure reminds me the most of functional programming practices which I have always personally had a hard time wrapping my head around.
   Personally, I have found Clojure quite confusing, harder to pick up than other languages we have looked at in this course. I understand the idea for absolute simplicity with this language but I think it goes a bit too far. At a certain point you lose the efficiency of trying to get simple. For example, not using commas to separate passed parameters to functions sacrifices readability at an attempt for simplicity. As the programmer it is a little harder to keep track of going on without the commas, personally I do not think its worth it. I would compare it to Python in its attempt to be simple, no semicolons and the use of tabs instead of brackets. However, Python maintains elements that might be 'superfluous' like commas to maintain readability.