CSC 435 Spring 2014

Assignment 5 - Clojure, An Unexpected Journey

Due: Friday, April 4th by 11:59 p.m.
Grade: 60 points

Objective:
The goal of this assignment is for students to grow more familiar with Clojure and Lisp-style programming. By the end of this assignment, students should be familiar with the following concepts in a Clojure context:

- ➢ Data as code
- ➢ Anonymous functions and Bindings
- ➢ Recursion ("loop" and "recur")
- ➢ Lazy evaluation and infinite lists

Requirements:
Two of the three parts (excluding Part 3) and the bonus part 4 of this assignment will include both a researched written section and a programming section. You are welcome to discuss the answers in class as it is being done, but you will complete and submit this assignment individually.

For each part of this assignment, create a new file in the lein project src directory that will be set up in part 0. The name of the file will be provided in each part as you create them.

The written parts of the assignment can all be contained in a single word document to be submitted with the rest of the assignment.

When you are finished, compress all of the clojure files, along with the written section and script file, into a single zip file named LastA5.zip (where Last is your last name) to be submitted on Canvas (see deliverables section more more details on submission).

Details:
Part 0 - Setting Up (5 points)
Part 1 - Basic Syntax and Data Structures (20 points)
Part 2 - Using Defined, Anonymous, and Built-in Functions (25 points)
Part 3 - Recursion (10 points)
BONUS Part 4 - Lazy evaluation, infinite lists, and ranges (+5 points)

Part 0 - Setting Up (5 points)

For this project, the Clojure project manager titled “Leiningen” will be used. If you do not already have this installed, follow the instructions on the Clojure wiki page on Canvas for proper installation. Alternatively, navigate to the Leiningen website for operating system specific instructions such as installation through a package manager (i.e. apt-get, homebrew, yum, pacman) or through the Windows installer.

We will begin by making a lein project. Open your terminal and navigate to the directory in which you would like the project to be created in. Create your project (named whatever you would like) and enter the project directory using the following commands (shown in UNIX-style commands).

```
$ lein new assignment5
Generating a project called assignment5 based on the 'default' template.
To see other templates (app, lein plugin, etc), try `lein help new`.

$ cd assignment5/
$ ls
LICENSE              doc           resources     test
README.md     project.clj    src
```

Next, we will make the file for the that will be used in part 1 of this assignment: “company.clj”. Once you do that, use your favorite text editor to assign a named reference to a data location. The steps are written out for you below.

```
$ touch src/company.clj
$ nano src/company.clj

(def hello "Hello Middle Earth")
```

With our our file defined, it is time to jump into the REPL and load in the program. Once we are in the REPL, we can load our file and call our reference to hello.

```
$ lein repl
nREPL server started on port 59301 on host 127.0.0.1
REPL-y 0.3.0
Clojure 1.5.1
    ...
user=> (load "company")
nil
user=> hello
"Hello Middle Earth"
```

Part 1 - Basic Syntax and Data Structures (25 points)

Congratulations, you have successfully entered Middle Earth by loading in your first file into the REPL. At this point, you may be asking yourself: "Great. I defined a string. What's the point of Clojure?" and "What exactly is a REPL?" Hopefully, answering these next few questions will help alleviate some of your built up tension before you get into some more programming in the language.

1.1 Clojure is said to be a dynamic programming language. Explain what that means and how it is different than a traditional object-oriented language such as Java.

1.2 We know that the dynamic nature of Clojure is implemented with the REPL. What does REPL stand for? How does it handle code that is typed or loaded into its interface?

1.3 Clojure compiles to Java byte code and runs on the JVM. For that reason, it is inherently linked to Java. One aspect of the language where it really shows its Java genes is in its data types and structures. To see what I mean, list seven non-collection data types used in Clojure and call the class method on them to see what it returns. I'll start you off with the first two more common types.

Data Type	Example	Class Call	Result of Class Call
Long (Integer)	1456793793	`(class 1456793793)`	java.lang.Long
String	"Frodo Baggins"	`(class "Frodo Baggins")`	java.lang.String

1.4 The previous question tasks you with listing non-collection data types in Clojure. This clearly implies that there are multiple collection types in Clojure, also called sequences. The four main collections are lists, vectors, sets, and maps. Give a brief description of each and provide an example of how one can be defined in Clojure. Again, I will give the the first one.

1. List - lists are used for code and evaluate as functions
   a. `(list 1 2 3 4)`
   b. `'(1 2 3 4)`

1.5 Before we get started with some code, there is one final thing you should know. If you

read the given definition of a Clojure list, you might be wondering what it means that lists evaluate as functions. This occurs because Clojure takes advantage of a feature called "data as code." Explain what this means as it relates to Clojure and where the language inherited this concept from.

Part 1 Programming Challenges

Type your answers to these questions in the `company.clj` file defined in Part 0 of this assignment.

1. **Create a vector named "dwarves" with the the names of the 13 dwarves featured in Tolkien's "The Hobbit." They are: Thorin, Balin, Bifur, Bofur, Bombur, Dori, Dwalin, Fili, Gloin, Kili, Nori, Oin, and Ori. The data type you use is up to you.**

2. **Create two references, "leader" and "company", where leader uses a collection operation to return the first element of the dwarves vector and company returns all elements except the first.**

3. **In the novel, Thorin's company recruits Bilbo to join their journey as a Burglar. Using a sequence operation, define a new reference called "full_company" which uses a collection operation on dwarves to add Bilbo to the collection.**

4. ***The first three questions were designed to get you started. These next two, however, require some more thought and perhaps a bit of research on functions built into Clojure.*** **First, define a vector of maps called "locations" to represent the information in the table below.**

5. **Use a built-in function or a list comprehension to define a reference called "warm_locations" that selects (filters) the names of all locations with an average temperature above 70 degrees fahrenheit (Hint - this question asks for the names of the locations, but try starting out by finding the maps with temperature > 70 first).***

Location Name	Average Temperature (F)
The Shire	74
The Misty Mountains	12
Fangorn Forest	58
Mordor	117
Rivendell	71

*Note - if you initially struggle with this problem, try moving on to Part 2, where functions will be discussed in detail, and then returning to this problem.

Part 2 - Using Defined, Anonymous, and Built-in Functions (25 points)

Now that you've learned about the basics of Clojure, let's move on to the real magic: functions.

First, we'll do a quick review of functions:

- Syntax: `(defn name "doc" [args] body)`
- `(def name body)` can be used to bind a name to a data structure.
- To see a function's documentation: `(doc function)`
- To see a function's source: `(source function)`

2.1 Four fundamental operators within Clojure include `map`, `filter`, `apply`, and `reduce`. Explain what each of these functions does.

2.2 Two methods of binding in Clojure include the use of `let` and `binding`. Explain the difference between the two.

2.3 Anonymous functions, or functions that are defined without being bound to an identifier, are commonly used in functional programming. Using similar format as shown above with the given syntax of a function, write the syntax of an anonymous function in Clojure.

2.4 A list comprehension is used to create a list out of an existing list. In Clojure, `for` takes a binding-form vector and a collection-expression pair to produce a list. Write the output for the following list comprehension:

```
(for [x [0 1 2 3 4]
                :let [y (* x 3)]
                :when (< y 7)]
          y)
```

Part 2 Programming Challenges

Write your answers in a new file, *wizardry.clj*.
Some of these challenges will use the `fellowship` list for data, so be sure to include the following data structure in your wizardly.clj file:

```
(def fellowship [{:name "Frodo", :race "Hobbit"} {:name "Sam", :race "Hobbit"}
{:name "Pippin", :race "Hobbit"} {:name "Merry", :race"Hobbit"} {:name "Gandalf",
:race "Wizard"} {:name "Aragorn", :race "Man"} {:name "Boromir", :race "Man"},
{:name "Gimli", :race"Dwarf"}, {:name "Legolas", :race "Elf"}])
```

1. Create a list of the lengths of the names of the members of the company defined in Part 1. (Since the company is just a list of names, this can be reworded as "create a list of the lengths of the elements in `company`.")

2. Take the list of lengths and add them together to find the sum of the company. This is

how tall the dwarves (and Bilbo) are when they stand on each other's shoulders.

3. Make a list of all the members of `fellowship` whose names are longer than 5 characters. (You don't have to list just the names -- keep the name and race together.)
   The first three exercises should have whetted your appetite. The following challenge is a little more challenging!

4. Repeat problem 3 put list only the names of the members, not the entire map

5. The fellowship needs to disguise itself! To do so, they will assume fake names. Gandalf suggests taking their real names and translating them into the ancient, cursed language of Pig Latin. **Using the `fellowship` list, create a new `fellowship` with all the names changed to the form: *estOfName-Ray*, e.g. "rodo-Fay".** For extra optional elegance, make the names properly capitalized, e.g. "Rodo-fay". **Make sure you preserve the structure of the original Fellowship!** {:name "Name", :race "Race"}
   Hint: If you get stuck on how you can use list functions to complete this, try using the following function, which returns a character(s) as a string

```
(defn makeStr "Takes a sequence of characters and returns a string" [chars] (apply str chars))
```

Part 3 - Recursion (10 points)

Your company has been scattered. As you trace through the Mines of Recursion, your company comes across an door with an ancient inscription. You must solve this puzzle quickly, before the pack of Orcs catches you. This door is the key to meeting the others at the three-towers. **Convert the following code to properly utilize the loop special form:**

Name this file mines.clj

```
(defn three-towers [n a b c]
(if (= n 1)
 (println (format "Move disk from %s to %s" a b))
   (do (three-towers (dec n) a c b)
        (println (format "Move disk from %s to %s" a b))
 (recur (dec n) c b a))))
```

(Bonus) Part 4 - Lazy evaluation, infinite lists, and ranges (+5 points)

I see you have made it through the Mines Of Recursion. Good job. Perhaps you should consider taking a rest in Lothlórien and visit Galadriel and Celeborn, because the next part of our journey will have have you walking in circles in the seemingly infinite forest with ranges upon ranges of trees every way you look. Don't be afraid of the forest, however; these next few questions will help you gain the experience you need to get through it. Should you fail, there are no consequences, as this is the bonus section of our journey. However, should you succeed. You shall be rewarded for doing so. For this part, create a new file and name it "roots.clj"

4.1 Like many other functional programming languages we have seen, Clojure takes advantage of a concept called lazy evaluation. Explain what this means and what advantages it provides for a language that implements it.

4.2 Without giving away too much of the answer to 4.2, having lazy evaluation in place makes having infinitely long lists and ranges theoretically possible. To generate our own [possibly infinitely long] lists, the functions range, repeat, iterate, and cycle can be used. Look up the documentation for each of these functions and give an example of the syntax they require to generate a list (Hint - you may also want to look up the "take" function to avoid an infinite loop problem).

Part 4 Programming Challenge

Fangorn Forest is full of tree-like creatures called Ents. They are the protectors of the forest, and have been in Middle Earth for as long as the elves have. For that reason, they are very, very old and their roots have grown long and numerous over the years. To keep track of all of their roots, the Ents have to to apply modern mathematical root finding methods to constantly calculate the location of their own roots. Some of these methods involve breaking up a polynomial equation into the coefficients associated with the even factors and the coefficients associated with the odd factors. For example, the polynomial $5x^4 + 10x^3 + 20x^2 + 50x + 100$ has the even coefficients (100, 20, 5) where 100 is the coefficient of the 0 exponential, 20 is the second exponential, and 5 is the fourth. The odd coefficients are (50, 10).

To help the ents find their roots, write two functions, "even_factors" and "odd_factors", that accept a list of roots (in order of decreasing exponential) and returns a list of the even/odd factors in order of increasing exponential. You will have to call upon the techniques you used in the previous sections in addition to your newly acquired knowledge of ranges and list generation to solve this problem. *(Hint - your initial thought may be to reverse the input, but you may not have to do that if you use list generation well!)* The input to this function is:

```
(def roots  [1 2 4 10 24 50 100 150])
```

Correct answer: even factors: (150, 50, 10, 2), odd factors: (100, 24, 4, 1)

Deliverables:
Submit on Canvas in the “Assignment 5” category a zip file containing:

- ➢ Source code files
- ➢ A script showing the correct loading-in and execution of your clojure code. This can be a single file for all parts.
- ➢ The written section in word-document format
- ➢ A readme.txt (text) file detailing the parts of of your project that do and do not function properly (organized by part). Also, clearly indicate whether you have attempted and/or completed the bonus part of the project

Grade:
As per the rubric.