Lab session 8 – Lisp/Clojure (part 1)

Unit	Programming languages: principles and design (6G6Z1110) Programming languages – SE frameworks (6G6Z1115)
Lecturer	Rob Frampton
Week	9
Portfolio element	This lab is not part of the portfolio.

Description

The aim of this lab sheet is to practise the basic Lisp/Clojure elements as studied in the lecture "Lisp/Clojure – part 1".

Note: This lab is meant to be performed on *repl.it* (https://repl.it/). On repl.it, you can create Clojure programs and compile them using a remote JVM. You can also create an account on *repl.it*, which is free, then select "*New Repl*" under the "*my repls*" menu and then, "*Clojure*" under the "*Practical*" programming languages section. Although the use of repl.it is very intuitive, take few minutes to get familiar with its GUI before you start with the lab exercises.

Exercises

Exercise 1a

Write the "*Hello, World!*" program in Clojure.

Exercise 1b

Change your program to display the following output:

```
12 squared is 144
```

where the value of 144 is computed by multiplying 12 by itself. You will need the `str` function.

Exercise 1c

Write a function called `square` that returns the square of its argument, which can be used like so:

```
> (square 4)
=> 16
```

and modify your program so that the `square` function is used.

Exercise 2

Create a function named `fact` which takes a single argument called *x* like so:

```
(defn fact [x]
    <code will go here>
)
```

This function should compute the factorial of the input x. You can do this in the following steps:

a) Generate a list of numbers from 2 up to x inclusive using the `range` function. Note that the upper argument to `range` is *exclusive* so you will need to add 1 to *x*. The `inc` function may be useful for this.
b) Multiply the list together using the `reduce` function. The arguments to `reduce` will be the list of numbers created above and the multiplication function.

You can test your code with the following:

```
   (fact 5)
=> 120
   (fact 6)
=> 720
```

Exercise 3

a) Write a method named `predict-balance` which takes three arguments: `initial`, `interestRate`, and `years`. It should return the balance of a savings account using the following formula:

$$forecast = balance * (1 + interestRate)^{years}$$

For example:

```
   (predict-balance 300 0.005 10)
=> 315.3420396122369
```

Hint: you can use Java's Math functions by using the Math namespace in your function names, for example: Math/pow

b) Write a method named `years-to-target` which takes three arguments: `initial`, `target` and `interestRate`. It should return the number of years required to reach the target amount using the following formula:

$$years = \frac{\log(target) - \log(initial)}{\log(1 + interestRate)}$$

Your function **should round up to the nearest year** before returning. For example:

```
   (years-to-target 300 400 0.005)
=> 58.0
```

Hint: Again, use Java's Math functions such as Math/log *and* Math/ceil

c) Write a function named target-years which takes three arguments: `initial`, `target` and `interestRate`. It should return a sequence of numbers from 0 to *n*, where *n* is the value given by `years-to-target`.

```
   (target-years 100 105 0.005)
=> (0 1 2 3 4 5 6 7 8 9 10)
```

Hint: you will need to use the `range` function. Don't forget that the upper argument to `range` is exclusive.

d) Write a function named `print-target` which takes three arguments: `initial`, `target` and `interestRate`. It should print out the predicted balance for each of the years given by `target-years`, as in the following example:

```
   (print-target 100 105 0.005)

Year 0: 100.0
Year 1: 100.49999999999999
Year 2: 101.00249999999997
Year 3: 101.50751249999996
Year 4: 102.01505006249995
Year 5: 102.52512531281243
Year 6: 103.03775093937651
Year 7: 103.55293969407337
Year 8: 104.07070439254373
Year 9: 104.59105791450642
Year 10: 105.11401320407896
=> nil
```

Use your `target-years` function to iterate over, and your `predict-balance` function to produce the yearly predictions.

Hint: you will need to use the `doseq` function, the `println` function and the `str` function.