Homework Assignment 2: Functional Programming

athena.ecs.csus.edu/~mei/135/hw/hw2.htm

Answers to this assignment should be submitted at Canvas as two files:

file 1 -- Put all solution definitions in (2) into one text file with your name as file name and with extension rkt ready to run in DrRaket;

file 2 -- prepare 2 testing cases for each problem solution definition above and show their interactive outputs in a document saved as pdf .

We are using DrRacket (HtDP book | download | Racket Guide) in our FL exercises, you should install the software first on your computer and then do the following exercises. Please choose Intermediate Student Level:

(1) Exercise Problems from the online book, Scheme tutorial by Takafumi Shido: Note: not every solution given in the online book will work in DrRacket due to the difference in implementation. The problems listed below should work and instructor has tried in DrRacket.

The following selected exercise-solution pairs from section 2-8 can be helpful to you to get ready for exercises in (2) and (3):

2. Using Scheme as a calculator: Ex1. #4, Ex2. #3

3. Making Lists: Ex2. #3, 5

4. Defining Functions: Ex1. #1

5. Branching: Ex1. #1, Ex2. #1, Ex3. #1

7. Repetition: Ex1. #1, 2

8. Higher Order Functions: Ex1 #1, 2

Note: There is no need to turn in the above exercises and they are for your practice only. You should read the tutorial and do the suggested exercise using DrRacket before working on the following problems in (2). There are three more online resources, one is developed by one of our graduate students and others are recommended by previous CSC 135 students. They can be helpful to you during this learning process:

(a) A courseware on Functional Programming -- this is developed for CSC 135 students who are learning functional programming using DrRacket. A set of examples/exercises can help you complete the assignment and learning basic concepts of functional programming.

(b) http://www.youtube.com/playlist?list=PLD0EB7BC8D7CF739A DrRacket Videos -- online resources recommented by CSC 135 students.

(2) For each problem 1-6, give a function definition with required function name and then run it with at least 2 testing cases using DrRacket:

1. The volume of a sphere of radius r is

volume = 4 * 3.14 * r^3 / 3

Write a function **sphere-volume** that takes one argument, radius, and returns the volume of the sphere with that radius. Then write a new function, shell-volume, that takes two arguments, inner-radius and outer-radius, and finds the volume of a hollow spherical shell with the appropriate inner and outer radii. (What we want is the volume between the two spheres.)

2. Write a predicate function **close?** which takes two numbers as arguments and returns #t if they are within, say, .001 of each other. You then extend this to write a new version of close? which takes three arguments -- number-1, number-2, and limit -- and returns #t if the two numbers are within a distance of limit from each other.

3. An equation is a claim about numbers; a quadratic equation is a special kind of equation. All quadratic equations (in one variable) have the following general shape:

In a specific equation, a, b and c are replaced by numbers, as in

$$a \cdot x^2 + b \cdot x + c = 0 .$$

The number of solutions for a quadratic equation depends on the values of a, b, and c. If the coefficient a is 0, we say the equation is degenerate and do not consider how many solutions it has. Assuming a is not 0, the equation has

$$2 \cdot x^2 + 4 \cdot x + 2 = 0$$

1. two solutions if b*b > 4 * a* c,
2. one solution if b*b = 4 *a * c, and
3. no solution if b*b < 4 * a * c.

Develop the function **how-many**, which consumes the coefficients a, b, and c of a proper quadratic equation and determines how many solutions the equation has:

```
(how-many 1 0 -1) = 2
(how-many 2 4 2) = 1
```

4. Write a function **filter-out-symbol** that takes as argument a list and a symbol and turns a list whose elements are the same as the argument list except with all instances of the symbol deleted. For example, evaluating

```
(filter-out-symbol '(no no a thousand times no) 'no)
```

should return the list (a thousand times).

5. [Louden 11.8 (b)]Write a function that computes the maximum and minimum of a list of integers. Required function name (**pMinMax** L).

6. [Louden 11. 8 (i)] Write a higher-order function inc_n that takes an integer n as a parameter and returns an n-th increment function, which increments its parameter by n. Thus in Scheme syntax, ((inc_n 3) 2) =5 and ((inc_n -2) 3) =1. Required function name: (**incnth** n)