Comp 311

Homework 1

Due: 11:59pm, Thursday, Sep 04, 2023

100 points

For all Racket assignments in this course, set the DrRacket Language to `Intermediate Student with lambda` (`Language → Choose Language → Teaching Languages → How to Design Programs → Intermediate Student with Lambda`). Your assignment will be tested using the specified language level. If you use a different language level, your code may not work when it is tested.

You will conduct and (implicitly submit) this assignment using GitHub Classroom. You will receive an invitation by email from GitHub Classroom within a few hours. Follow the link to a personal GitHub repository containing this assignment.

- Carefully follow the **Sample Solution to a Programming Problem** in the **Racket HW Guide**. Only half of the credit for each programming problem is based on the correctness of your code as determined by our test cases. Much of the grade is based on how well you follow the *design recipe*. For a crisp example of an ideal solution, look at the **Sample Solution to a Programming Problem** that sorts **list-of-number** at the end of **Sample Solution to a Programming Problem** in the **Racket HW Guide**. This process may appear painful but it shows "in the small" how program design should be done "in the large". It is not difficult. If you carefully inspect the sample program (notably the commenting), it shows the level of documentation of your program *design* that we expect.
- **Do the following programming problems using only the primitives mentioned in Lectures 2 and 3. Do not use the functions in the Racket library if they are not mentioned in Lectures 2 and 3.**

  1. [10 pts] Develop the function `contains?` that consumes a symbol and a list of symbols and determines whether or not the symbol occurs in the list.

  2. [10 pts] Develop the function `count-symbols` that consumes a list of symbols and produces the number of items in the list.

  3. [10 pts] Develop the function `count-numbers` that counts how many numbers are in a list of numbers. [Note: the function merely works on inputs that are lists of numbers; it may blow up on anything else].

  4. [20 pts] Develop the function `avg-price`. It consumes a list of toy prices and computes the average price of a toy. The average is the total of all prices divided by the number of toys. Toy prices are numbers.

     Hints:

- Develop one or more auxiliary functions (following the design recipe for each auxiliary function) to make the definition of `avg-price` easy. Do not use the Racket library other than primitives mentioned in Lectures 2 and 3.
- If the list of toy prices is empty, the function `avg-price` generates an error report as described in Guidance below.

5. [10pts] Develop the function `elim-exp` to eliminate expensive toys. The function consumes a number, called `mp` (short for "maximum price") and a list of toy prices, called `lotp`, and produces a list of all those prices in `lotp` that are below or equal to `mp`. For example

   ```
   (check-expect (elim-exp 1.0 (list 2.95 .95 1.0 5) (list .95 1.0)) = #true
   ```

6. [10pts] Develop the function `delete` to eliminate specific toys from a list. The function consumes the name of a toy, called `ty`, and a list of names, called `lon`, and produces a list of names that contains all components of `lon` with the exception of `ty`. For example,

   ```
   (check-expect (delete 'robot (list 'robot 'doll 'dress)) (list 'doll 'dress))
   = #true
   ```

7. [30pts] A list can be used to represent a finite set. For example,

   ```
     (list 'c 'o 'm 'p)
   ```

   represents the set of symbols {`'c` , `'o`, `'m`, `'p`}. In such a representation, we assume all elements in the list are unique; there are no duplicates.

   Develop the function `power` that consumes a list of symbols `los` (representing a set) and produces a list of list of symbols representing the power set (set of all subsets) of `los`. **Hint:** write an auxiliary function `cons-all` that consumes a symbol `sym` and a list of list of symbols `lolos` and inserts symbol `sym` at the front of each list in `lolos`.

   For example,

   ```
     (check-expect (cons-all 'a (list (list 'c) (list 'o) (list 'm) (list 'p))

                   (list (list 'a 'c) (list 'a 'o) (list 'a 'm) (list 'a 'p)))
    = #true
   ```

- **Guidance:**

1. Follow the design recipe imitating **Sample Solution to a Programming Problem** in the **Racket HW Guide.**

2. Problem 4 asks you to write a function that checks for the empty list as an input error and reports an aborting error in the case. (The purpose statement should document this behavior!) In DrRacket, the `error` function takes a single argument, a string specifying

the error message to report, *not two arguments* as described in the book. The DrRacket Help Desk correctly documents the Racket `error` function. We recommend using a string (text enclosed in double quotation marks) like `"An empty list of toy prices is an illegal input."` as the argument. You can test the error-throwing behavior of a function using the `check-error` operation which is documented in the DrRacket Help Desk.

3. Study Figure 26 in Section 9.4 of the book for a more detailed description of the design recipe and how it should be documented in the program text that you develop.

4. To follow the design recipe, you must:

   - write the *type contract* and *purpose* (behavioral contract);
   - provide at least 3 well-chosen examples (more for complex functions like `power`);
   - write the template for the function (trivial when no recursion is involved);
   - write the code for the function; and
   - include illustrative test cases for the function, using at least the examples you developed above.

-