CS115 **Assignment 05** Fall 2019

Due: Wednesday, October 23 @ 10:00 AM (No late submissions)

Assignment Guidelines:

- Solutions to these questions are expected to follow the requirements of the Style Guide (https://www.student.cs.uwaterloo.ca/~cs115/coursenotes1/styleguide.pdf). This includes all relevant design recipe elements, proper use of constants, and proper use of helper functions.
- Submission details:
  - Solutions to these questions must be placed in files **`a05q1.rkt`**, **`a05q2.rkt`**, **`a05q3.rkt`**, respectively, and must be completed in Racket.
  - All solutions must be submitted through MarkUs. Solutions will **not** be accepted through email.
  - Verify your basic test results using MarkUs to ensure that your files were submitted properly and are readable on MarkUs. *Note, however, that passing the basic tests does not guarantee that you will pass all our correctness tests.*
- Download the interface file from the course Web page to ensure that all function names are spelled correctly, and each function has the correct number and order of parameters.
- Restrictions:
  - You may only use the built-in functions and special forms introduced in the lecture slides up to and including the module covered by this assignment. A list of these functions can be found on the Assignments web page: https://www.student.cs.uwaterloo.ca/~cs115/#allowed
  - Read each question carefully to see if any additional restrictions apply.
  - Test data for correctness tests will always meet the stated assumptions for consumed values.
- The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.

Plagiarism: The following applies to all assignments in CS115.

All work in CS 115 is to be done individually. The penalty for plagiarism on assignments (first offense) is a mark of 0 on the affected question and 5 marks off the final grade, consistent with School of Computer Science policy. In addition, a letter detailing the offense is sent to the Associate Dean of Undergraduate Studies, meaning that subsequent offenses will carry more severe penalties, up to suspension or expulsion.

To avoid inadvertently incurring this penalty, you should discuss assignment issues with other students only in a very broad and high-level fashion. Do not take notes during such discussions, and avoid looking at anyone else's code, on screen or on paper. If you find yourself stuck, contact the ISA or instructor for help, instead of getting the solution from someone else. Do not consult other books, library materials, Internet sources, or solutions (yours or other people's) from other courses or other terms.

Read more course policies at: https://www.student.cs.uwaterloo.ca/~cs115/#policies

Language level: Beginning Student
Coverage: Module 4

Due: Wednesday, October 23 @ 10:00 AM (No late submissions)

Question 1: Judging Gymnastics

In most official gymnastics competitions, including at the Olympics, a panel of judges determines the "Execution Score" for a routine. Each judge independently computes a score, which is a natural number between 0 and 10 (inclusive). The highest and lowest of the scores are then dropped to prevent individual judges from biasing the outcome, and the Execution Score is calculated as the average of the remaining values.

Complete a Racket function **`execution-score`** that consumes **`scores`**, a non-empty list of natural numbers, all between 0 and 10, and produces the correct execution score according to the rules above. For example:

```
(execution-score (cons 6 (cons 5 (cons 9 (cons 2 empty)))))
   ⇒ 5.5

(execution-score (cons 10 (cons 1 (cons 10 (cons 4 (cons 7
empty))))))
   ⇒ 7
```

Note that exactly one highest and one lowest score should be removed from the list, even if there are ties (in the second example, the final score is the average of 10, 4, and 7—only one of the 10s is removed). You can assume there are at least three scores in the initial list.

Submit your solution in the file **`a05q1.rkt`**.

Question 2: ROT-13

ROT-13 is a simple scheme for encoding text. It's not meant to be difficult to crack, only difficult to read when encoded. Decades ago, it was used to post spoilers and other sensitive information on the internet, allowing readers to choose if they wanted to see the decoded text.

The encoding method is simple. Imagine listing the letters of the alphabet in a circle. Now, given an input message, replace every letter by the one found exactly 13 spaces later around the circle. For example, "c" becomes "p" and "s" becomes "f".

Complete the Racket function **`rot-13`** that consumes a string **`msg`** and produces a new string in which all the letters have been transformed as described. You should process the string using the recursive approach described in class, converting to and from a list of characters with **`string->list`** and **`list->string`**. You must correctly handle both lowercase and uppercase letters, and leave any non-letter characters unchanged. So, for example,

```
(rot-13 "Computer Science 115") ⇒ "Pbzchgre Fpvrapr 115"
```

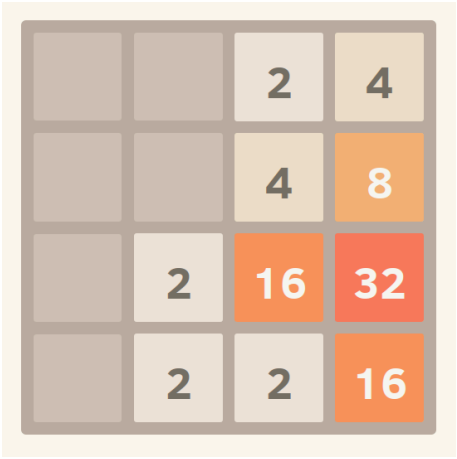
You'll want to write a helper function that converts a single character. Don't write a **`cond`** with a case for every possible letter! Instead, use arithmetic on the numerical equivalents of the letters. You may use the built-in functions **`char->integer`** and **`integer->char`**. For example, if **`ch`** holds a lowercase letter, then

```
(- (char->integer ch) (char->integer #\a))
```

will produce a natural number from 0 to 25, corresponding to the letters a through z in order.

Submit your solution in the file **`a05q2.rkt`**.

Question 3: 2048



2048 is an addictive browser-based puzzle in which you manipulate a grid of tiles marked with powers of two. When tiles containing the same number collide, they combine to form a single tile of the next higher power of two. The goal is to produce the 2048 tile. The original game took place on a 4×4 grid, but it can be played on a grid of any size.

In each move, you shift the tiles to the left, right, up, or down, using the arrow keys. What happens as a result is described by this excerpt from Wikipedia's page:

> Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move.
>
> If a move causes three consecutive tiles of the same value to slide together, only the two tiles farthest along the direction of motion will combine. If all four spaces in a row or column are filled with tiles of the same value, a move parallel to that row/column will combine the first two and last two.

Let's focus on a single row of the grid, represented as a list of integers, with zeros denoting empty cells. We will also assume that we're only sliding tiles to the left (the equivalent of pressing the left arrow in the actual game).

Complete the Racket function **`slide-left`**, which consumes **`lon`**, a list of natural numbers of any length, and produces a new list of natural numbers of the same length, containing the

values that result from sliding tiles to the left according to the rules explained above. One possible way to break down the problem is as follows:

1. Remove all the zeros from the given list, producing a new list that might be shorter than the original.
2. Walk over the list created in Step 1, producing a new list in which consecutive identical values are combined (added). Remember that if three consecutive tiles are the same, only the first two will be combined. *Note that this step may require you to combine the first and second elements of a non-empty list—it's likely that in this case, you'll need to apply your function recursively to* **`(rest (rest lst))`** *for some list* **`lst`**.
3. Add enough zeros onto the end of the list produced in Step 2 to restore it to the length of the original list. You are permitted to use the built-in functions **`append`** and **`make-list`** for this step.

Here are some illustrative examples:

```
(slide-left empty) ⇒ empty

(slide-left (cons 1024 (cons 0 (cons 1024 empty))))
    ⇒ (cons 2048 (cons 0 (cons 0 empty)))

(slide-left (cons 4 (cons 4 (cons 4 (cons 1 (cons 1 empty))))))
    ⇒ (cons 8 (cons 4 (cons 2 (cons 0 (cons 0 empty)))))

(slide-left (cons 8 (cons 8 (cons 8 (cons 8 empty)))))
    ⇒ (cons 16 (cons 16 (cons 0 (cons 0 empty))))

(slide-left (cons 2 (cons 1 (cons 2 (cons 0 empty)))))
    ⇒ (cons 2 (cons 1 (cons 2 (cons 0 empty))))
```

Note that although non-empty tiles in the original game must contain powers of two starting at 2, your solution should work correctly on any natural numbers. For example:

```
(slide-left (cons 3 (cons 0 (cons 0 (cons 3 (cons 7 empty))))))
    ⇒ (cons 6 (cons 7 (cons 0 (cons 0 (cons 0 empty)))))
```

Submit your solution in the file **`a05q3.rkt`**.