- For this and all subsequent assignments, you are expected to use the design recipe when writing functions from scratch, including helper functions.
- **For full marks, it is not sufficient to have a correct program. Be sure to follow all the steps of the design recipe. Read the Style Guide carefully to ensure that you are following the proper conventions.** ***In addition, your solution must include the definition of constants and helper functions where appropriate*****.**
- Unless otherwise indicated by the question, you may only use the built–in functions and special forms introduced in the lecture slides from CS115 up to and including the modules covered by this assignment.
- 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.
- Read each question carefully for restrictions.
- Test data for all questions will always meet the stated assumptions for consumed values.
- Do **not** copy the purpose directly from the assignment description. The purpose should be written in your own words and include references to the parameter names of your functions.
- 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.
- Do **not** send any code files by email to your instructors or tutors. Course staff will **not** accept it as an assignment submission. Course staff will **not** debug code emailed to them.
- You may post general assignment questions using the discussion groups on Waterloo LEARN. Choose Connect then Discussions. Read the guidelines for posting questions. Do NOT post any code as part of your questions.
- Check MarkUs and your basic test results to ensure that your files were properly submitted. In most cases, solutions that do not pass the basic tests will not receive any correctness marks.
- Read the course Web page for more information on assignment policies and how to organize and submit your work. Follow the instructions in the Style Guide. Your solutions should be placed in files `a5qY.rkt`, where `Y` is a value from 1 to 4.
- Since each file you submit will contain more than one function, **it is very important that your code runs**. If your code does not run then none of the functions can be tested for correctness.

Plagiarism: The following applies to all assignments in CS115.

- Be sure to read the Plagiarism section at:
  https://www.student.cs.uwaterloo.ca/ cs115/#assignments

Language level: Beginning Student
Coverage: Module 5 up to slide 58

Make sure to use constants and helper functions whenever appropriate.

1. Write a Racket predicate `shouting?` that consumes a string `st`, and produces `true` if the string contains more CAPITAL LETTERS than it does lower case letters. In all other cases it produces `false`. If the string contains no letters, the predicate produces `false` (because 0 is not more than 0).
   For example:
   - `(shouting? "HELLO WorlD!") => true`
   - `(shouting? "") => false`

2. Write a Racket function `replace-all` that consumes three parameters, a list of strings (`lst`), and two strings (`match` and `rep`). The function produces a new list with the same contents as `lst`, but with all occurrences of `match` (if any) replaced with `rep`.
   For example:
   - ```
     (replace-all
     (cons "I" (cons "like" (cons "CS115" empty))) "like" "LOVE")
     => (cons "I" (cons "LOVE" (cons "CS115" empty)))
     ```

3. Write a Racket function `filter-out` that consumes a non-empty list of integers, `loi`, and produces a new list with the same contents as `loi`, but without all the occurrences of the first integer in `loi`.
   For example:
   - ```
     (filter-out (cons 10 (cons 4 (cons 10 (cons 4 (cons -3 empty))))))
     => (cons 4 (cons 4 (cons -3 empty)))
     ```
   - `(filter-out (cons 4 (cons 4 (cons 4 empty)))) => empty`

4. Useful structures and data definitions

```
(define-struct student-grades (username numgrades))
;; A Student-Grades is a (make-student-grades Str (listof Int))
;; requires:
;; numbers are between 0 and 100, inclusive

(define-struct student-results (username lettgrades))
;; A Student-Results is a (make-student-results Str (listof Str))
;; requires:
;; Str in the list is any of "A" "B" "C" "D" "F"
```

Write a Racket function `convert` that consumes a `student-grades` value and produces a `student-results` value with the same content while using the following grading system:

Letter Grade	Numerical Grade
A	90-100
B	80-89
C	70-79
D	60-69
F	0-59

For example:

- ```
  (convert
  (make-student-grades
  "mx456"
  (cons 90 (cons 89 (cons 80 (cons 79 (cons 70 (cons 69 (cons
  60 (cons 59 (cons 50 empty)))))))))) =>
  (make-student-results
  "mx456"
  (cons "A" (cons "B" (cons "B" (cons "C" (cons "C" (cons "D"
  (cons "D" (cons "F" (cons "F" empty)))))))))
  ```
  ```