• For this assignment, you are NOT allowed to use recursions (in any of the functions you define including the helper functions), all questions should be solved using abstract list functions.

• All helper functions (if allowed) must be defined locally including purpose and contact, however you are encouraged to test them outside local expressions first if possible.

• For this assignment, you are expected to use the design recipe when writing the main 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 a8qY.rkt, where Y is a value from 1 to 2.

• 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.
1. For this question, using helper functions is allowed however should be defined **locally**. lambda is allowed as well.

(a) Write a Racket function `shouting` that consumes a string `st`, and produces `true` if the string contains more CAPITAL LETTERS than it does lower letters. In all other cases it produces `false`. If the string contains no letters, the function returns `false` (because 0 is not more than 0).

For example:

- `(shouting "HELLO World!")` => `true`
- `(shouting ")"`) => `false`

(b) 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 (list "I" "like" "CS115") "like" "LOVE")` => `(list "I" "LOVE" "CS115")`

(c) Write a Racket function `above-average` that consumes a non-empty list of integers (`lon`), and produces a new list of all integers from `lon` which are above the `lon` average.

For example:

- `(above-average (list 5))` => `empty`
- `(above-average (list 0 -100 20 6 -300 12 400 -5 20 6))` => `(list 20 6 12 400 20 6)`

(d) Write a Racket function `word-chain?` that consumes a list of strings (`los`), and produces `true` if the strings in `los` is a word chain where the last letter of each word is the first letter of the next word, otherwise the function produces `false`.

For example:

- `(word-chain? empty)` => `false`
- `(word-chain? (list "abc"))` => `true`
- `(word-chain? (list "abc" "cd" "dad"))` => `true`
2. For this question, using helper functions is NOT allowed except `lambda`.

(a) Write a Racket function `filter-out` that consumes a non-empty list of integers, `lon`, and produces a new list with the same contents as `lon`, but without all the occurrences of the first integer in the consumed list.
   You may NOT use `remove-all`.
   For example:
   • `(filter-out (list 10 4 10 4 3))` => `(list 4 4 3)`
   • `(filter-out (list 4 4 4))` => empty

(b) Write a Racket function `factorial` that consumes a natural number, `n`, and produces the factorial of `n`.
   Note: The factorial of `n` is the product of `1, 2, 3, ..., n`
   For example:
   • `(factorial 0)` => 1
   • `(factorial 5)` => 120 (1*2*3*4*5=120)

(c) Write a Racket function `max-diff` that consumes a non-empty list of natural numbers, `lon`, and produces the difference between the maximal and the minimal values in `lon`.
   For example:
   • `(max-diff (list 1 10 0 4 -5 4 0 ))` => 15
   • `(max-diff (list 4))` => 0

(d) Write a Racket function `avg-len` that consumes a non-empty list of strings, `los`, and produces the average of strings’ lengths in `los`.
   For example:
   • `(avg-len (list "nice" "job" "ha"))` => 3
   • `(avg-len (list "nice-job" "solving" "a8"))` => 5.6666...

(e) Write a Racket function `largest-sum-lists` that consumes a list of lists of integers, `lols`, and produces the maximal sum of the internal lists in `lols`.
   For example:
   • `(largest-sum-lists (list (list 1 2 3 ) empty (list 765)))` => 765
   • `(largest-sum-lists empty)` => 0

(f) Write a Racket function `in-range` that consumes a list of integers, `lon`, and two integers `a` and `b` where `a<=b`, and produces the number of values in `lon` that are between `a` and `b` inclusive.
   For example:
   • `(in-range empty 4 12)` => 0
   • `(in-range (list 34 700 56 2 -45 98 -100 -300 456 10 )-200 500)` => 8
(g) Write a Racket function `build-matrix` that consumes a positive natural number, `n`, and a string `st`, and produces a list of lists of `st` of size nXn. For example:

- `(build-matrix 1 "hi") => (list (list "hi"))
- `(build-matrix 3 "*") => (list (list "*" "*" "*") (list "*" "*" "*") (list "*" "*" "*"))`