Assignment Guidelines:
- 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 in the question you may use only the built-in functions and special forms introduced in the lecture slides from CS115 up to and including the modules covered by this assignment. A list of functions described in each module of the lecture slides can be found at https://www.student.cs.uwaterloo.ca/~cs115/built_in
- 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 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 a7qY.rkt, where Y is a value from 1 to 4.

Plagiarism: Read https://www.student.cs.uwaterloo.ca/~cs115/assignments#Plagiarism

Language level: Beginner Student with List Abbreviations
Coverage: Module 06 and 07
Question 1:
Assocation lists were introduced in Module 06. In this question, you will use a variation on the association list described in class to help the Registrar's Office manage room bookings.

;;; A Room Association (RA) is a (list Str Nat), in which
;;;   the first value is the classroom (the key) and
;;;   the second value is associated room capacity.

;;; A Room Association List (RAL) is one of
;;; * empty
;;; * (cons RA RAL)
;;; requires: all classrooms (keys) are distinct.

Complete the Racket function book-room that consumes available-rooms (a RAL) containing all available classrooms with their capacities, rm (a Str) for the classroom, and size-needed (a Nat). If rm is a key in available-rooms and its capacity is at least size-needed, then the function produces an RAL like available-rooms, but without rm's association. If the capacity of rm is not large enough, or rm is not actually a key in available-rooms, then the function returns false.

For example,

(define classrooms (list (list "MC 2038" 96) (list "DC 1350" 252)))
(book-room classrooms "DC 1350" 190) => (list (list "MC 2038" 96))
(book-room classrooms "DC 1350" 345) => false

Question 2:
How do we calculate your clicker grades? The first step is determining how many questions were answered (correctly or incorrectly) and how many were missed. In this question, you will write a function that consumes answers for one student, along with the correct answers, and produces a Clicker-Statistic value for the student:

(define-struct clicker-statistic (total correct incorrect unanswered))
;;; A Clicker-Statistic is a (make-clicker-statistic Nat Nat Nat Nat)),
;;; requires:
;;;   total = correct + incorrect + unanswered

Complete the Racket function clicker-results that consumes two lists of the same length (student and expected), and produces a Clicker-Statistic value with the student's results. The entries in the student list are symbols for the answers: 'A, 'B, 'C, 'D, 'E for answered questions, and 'X if the question was not answered. The entries in the expected list are themselves lists, containing all the correct answers for a question. For example, if 'A was the only correct answer to a question, then the corresponding entry in expected is (list 'A). However, if both 'C and 'D were worth full marks, then expected contains (list 'C 'D). Note that 'X will never be considered a correct answer. For example,

(clicker-results (list 'A 'C 'A 'D 'B 'E 'D 'X)
    (list (list 'A) (list 'B) (list 'A) (list 'A 'B 'C) (list 'D)
    (list 'A 'B 'C 'D 'E) (list 'E) (list 'A)))
=> (make-clicker-statistic 8 3 4 1)
Question 3
Complete the Racket function `nats-in-order?` that consumes a list of natural numbers (counters) and a single natural number (n), and produces `true` if the natural numbers 0,1,2, through n appear in increasing order in counters, and produces `false` otherwise. Note that the numbers 0,1,2,…,n do not need to appear in consecutive positions in counters, and counters may contain any Nat. For example,

(nats-in-order? (list 0 1 2 3 5) 3) => true
(nats-in-order? (list 5 0 1 2 3) 3) => true
(nats-in-order? (list 3 0 3 1 3 7 2 3) 3) => true (since 0,1,2,3 appear in increasing order in the list, with values in between them)
(nats-in-order? (list 4 3 2 1 0) 3) => false (while 0,1,2,3 appear in the list, they are not in increasing order)

Question 4
In this question, you will play a simplified version of the card game "War" described in Question 7 of Lab 08. You can complete this function without having completed that lab question.

(define-struct card (rank suit))
;;;; A Card is a (make-card Nat (anyof "hearts" "spades" "diamonds" "clubs"))
;;;; requires: 1 <= rank <= 13

(define-struct player (name hand))
;;;; a Player is a (make-player Str (listof Card))

In our card game, two players each have a collection of cards (called a hand). They do not need to have the same number of cards in their hands. The game is a series of battles, where players compare the next cards in their hands. The player whose card has a higher rank wins the battle. If the two cards have the same rank, then no one wins that battle. If one player runs out of cards before the other, the player with cards remaining wins a battle for each remaining card.

Complete the Racket function `card-game-winner` that consumes two Player values, and produces the name of the player who wins more battles. If the two players win the same number of battles, then the function produces the string "tie". For example,

(card-game-winner
 (make-player "Aly" (list (make-card 2 "spades") (make-card 8 "spades")
 (make-card 12 "hearts") (make-card 7 "diamonds")))
 (make-player "Bo" (list (make-card 11 "hearts") (make-card 8 "clubs")
 (make-card 4 "spades")))
) => "Aly"

In the example game,
- Bo wins the first battle (11 beats 2).
- The second battle is a tie (both 8).
- Aly wins the third battle (12 beats 4).
- Aly wins the fourth battle (since Bo only has three cards).

You can assume that there is no player named "tie", so there will not be any ambiguity with the result.