Assignment 06
Due: Wednesday, June 28th, 2017 at 10am

• If you write a solution that includes a wrapper function, you should provide the full design recipe, but wrapped functions require only the purpose and contract.
• You must provide the data definition and template in your solutions only when the question specifically indicates they are required for compound data types described in the question. If you create any additional data types that are beyond the question description, your program file should include a data definition and a template for each additional data type.
• If you include a template in your solution, the template should appear as comments.
• You may want to include defined constants to help reduce the writing for the examples and tests.
• 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. A list of functions described in each module of the lecture slides may be found at https://www.student.cs.uwaterloo.ca/~cs115/built_in.
• Use the design recipe when writing functions (and helper functions) from scratch.
• 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.
• You may post general assignment questions using the discussion forum on Waterloo LEARN. Choose Connect -> Discussions. Read the guidelines for posting questions. Do NOT post any code as part of your questions.
• 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.
• 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.
• Any string or symbol values must exactly match the descriptions in the questions. Any discrepancies in your solutions may lead to a severe loss of 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 a06qY.rkt, where Y is a value from 1 to 3.

Language level: Beginning Student with List Abbreviations
Coverage: Module 6

1. Write a function list-of-squares that consumes a natural number n and produces a list of perfect squares between 1 and n (inclusive) in descending order. If n is 0, the function should produce an empty list.

For example:
(list-of-squares 10) => (list 9 4 1)
2. An anagram is a string formed by rearranging the letters of another string. For example, “bat”, “abt”, “atb”, “tab” and “tba” are all anagrams of “tab” and “diet” is an anagram of “tied”. Write a function called anagram-list that consumes a list of strings los, and a non-empty string str, and produces a list that contains all anagrams of str in los.

For example:
(anagram-list (list "puppet" "diet" "edited" "tide" "tied") "tide") =>
(list "diet" "tide" "tied")

3. Use the following structures and data definitions to complete this question:
(define-struct team (name packages))
;;;; A Team is a (make-team Sym (listof Nat))
;;;; requires: packages is a non-empty list
;;;; all the numbers in packages > 0

(define chips (make-team 'chips (list 25 2 8 10 22 18)))
(define cookies (make-team 'cookies (list 10 23)))
(define candies (make-team 'candies (list 20)))
(define cereal (make-team 'cereal (list 2)))

a) Write a Racket function called total-packaged that consumes a list of teams and produces the total amount packaged by all teams in the list. For example:
(total-packaged (list chips cookies)) => 118

b) Write a Racket function called best-team that consumes a non-empty list of teams and produces the team with the highest amount of packages from the list. If there’s a tie, produce the first team in the list with the highest package.

For example:
(best-team (list cookies chips)) =>
(make-team 'chips (list 25 2 8 10 22 18))
(best-team (list candies)) => (make-team 'candies (list 20))

c) Modify the insert and sort functions in Module 06, Slides 41-47, into insert-team and sort-teams. The sort-teams function consumes a list of teams and produces a new list of teams sorted in a non-decreasing (ascending) order based on the total amount of packages they have. You may assume no two teams in the list packaged the same amount of items.

For example,
(sort-teams (list cookies candies chips cereal)) =>
(list cereal candies cookies chips)