Assignment 07
Due: Wednesday, July 5th, 2017 at 10am

• If you write a solution that includes a wrapper function, you are only required to include examples and tests for the wrapper function, and not for the function that it wraps.
• 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 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.
• 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 a07qY.rkt, where Y is a value from 1 to 3.

Language level: Beginning Student with List Abbreviations
Coverage: Module 7

1) Write a Racket function, destination that consumes a list of non-empty strings (representing names of Canadian cities) and a list of positive numbers (representing the corresponding prices of flight tickets to the cities). You may assume that both lists are of the same length and both are non-empty. The function should produce the name of the city corresponding to the cheapest flight; in case of a tie, produce the name of the first city in the list with the cheapest ticket cost.

For example:
(destination (list "calgary" "waterloo" "halifax" "winnipeg")
 (list 225.82 73.55 90.5 80)) => "waterloo"
2) In an exam with multiple-choice questions, a student is required to get at least a certain mark to pass. The correct answers for the questions are represented by the symbols 'A', 'B', 'C' and 'D'. A student's answers are also represented by the same set of symbols.

Write a Racket function `student-pass?` that consumes two non-empty lists of letter symbols of equal length, `corr-ans` (list of correct answers) and `stud-ans` (list of student's answers), and a natural number `pass-mark` and produces true if there are at least `pass-mark` correct matches. A correct match means symbols at the i\textsuperscript{th} index in both lists are the same, that is, the student's answer is correct. You can assume that `pass-mark` never exceeds the number of given questions, i.e. the length of `corr-ans` and `stud-ans`. Note that if `pass-mark` is 0, `student-pass?` always produces true, i.e. the student always passes.

For example:
```
(student-pass? (list 'A 'A 'C 'D 'B 'D) (list 'A 'C 'C 'D 'B 'D) 4) => true
(student-pass? (list 'C 'B 'A 'B) (list 'D 'B 'A 'B) 3) => true
(student-pass? (list 'D 'A 'D 'D 'C) (list 'D 'B 'C 'B 'A) 3) => false
```

3) Using structural recursion, write a function called `fill-in-blanks` that consumes a string `sentence` containing underscore characters (_), and a list of strings `alos`. The function produces a string with all of the _ characters replaced by a string in `alos` in the order in which they appear. You may assume that the number of _ occurrences in `sentence` matches the number of characters in `alos`.

For example:
```
(fill-in-blanks "Su_e_s_e_!" (list "mm" "r i" " gr" "at")) => "Summer is great!"
```