Assignment: 5

Due: Thursday, June 18 at 9:00 am
Language level: Beginning Student
Coverage: Modules 1–5

For this and all subsequent assignments, to receive full marks you are required to use the design recipe for every function you write. You may include the examples given in the assignment in your submissions, but they will be ignored by the markers—you must develop a complete suite of examples and tests on your own. For your convenience, an interface file which contains the headers of the required functions is available on the course webpage.

Do not send any code files to course staff; they will not be accepted. Submissions must be made via MarkUs as described on the course webpage. After submission, check your basic test results to ensure your files were properly submitted. Solutions that do not pass the basic tests are unlikely to receive any correctness marks.

Remember, the solutions you submit must be **entirely your own work**.

1. The recording of transactions is one important application for which lists are commonly used. In this question, you will write a function which processes a list of credit card purchases and produces the total charge on a credit card bill with those purchases. The cost of each purchase will be given before tax and fees; there will be a 13% sales tax on each purchase as well as a 25 cent credit card fee which is charged by the merchant on every purchase (after taxes, so no tax is paid on the fee).

   Furthermore, this question will make use of the following data definition:
   ;; A DollarAmount is a nonnegative number with at most two decimal places

   Write a Racket function *credit-card-bill* which consumes a list of *DollarAmounts* and produces a *DollarAmount* which represents the total charge on a credit card bill containing the purchases in the given list, including tax and fees as described above.

   The amount of tax charged on each purchase should always be rounded to the nearest cent so that the charge including tax is still a *DollarAmount*. For example, the tax on a 99 cent purchase should be 13 cents. The built-in function *round* could be useful here, but it rounds to the nearest integer, not the nearest cent—a hint that you might want to define your own rounding function!

   Example: (*credit-card-bill* (*cons* 3.50 (*cons* 2.99 (*cons* 5.00 *empty*)))) should produce 13.74, since there will be a charge of \$11.49 in purchases, $\$0.46 + \$0.39 + \$0.65$ in tax, and \$0.75 in fees.

   Note: Your function must handle the empty list. Due to the way recursion works, *every* application of *credit-card-bill* will ultimately rely on this case being handled correctly.

2. (a) Write a Racket function *comes-first* which consumes two symbols and a list of symbols (in that order) and produces which of the two given symbols occur first in the given list. If neither of the two given symbols occurs in the given list, the function should produce *false*.

   For example, (*comes-first* 'a 'b (*cons* 'c (*cons* 'a (*cons* 'b *empty*)))) should produce 'a and (*comes-first* 'a 'b (*cons* 'c (*cons* 'd (*cons* 'e *empty*)))) should produce *false*.

   (b) Write a Racket function *swap-symbols* which consumes two symbols and a list of symbols (in that order) and produces a list which contains the same symbols as the given list except that all instances of the first given symbol have been replaced with the second given symbol, and vice versa.

   For example, (*swap-symbols* 'a 'b (*cons* 'c (*cons* 'a (*cons* 'b *empty*)))) should produce (*cons* 'c (*cons* 'b (*cons* 'a *empty*))).

3. Write a Racket function *initials* which consumes a nonempty string containing the full name of a person or institute and produces a string containing the initials of that person or institute. For example, (*initials* "Jonathan William Smith") should produce "JWS".

   Formally, an initial is either the very first letter of a string or appears immediately after a space. There may be more than three initials, and not every initial may be capitalized. For example, (*initials* "Ludwig von Mises Institute") should produce "LvMI".

   You may assume that the given string contains only alphabetic characters and spaces, that single spaces are used to separate names, and that spaces will not occur anywhere else in the string.

   Hint: To translate a string into a list of characters for processing you should make use of the built-in Racket function *string→list*. Additionally, the function *list→string* allows you to perform the opposite translation.