CS115 **Assignment 09** Fall 2019

Due: Wednesday, November 27 @ 10:00 AM (No late submissions)

Assignment Guidelines:

- Solutions to these questions are expected to follow the requirements of the Style Guide (https://www.student.cs.uwaterloo.ca/~cs115/coursenotes1/styleguide.pdf) This includes all relevant design recipe elements, proper use of constants, and proper use of helper functions.
- Submission details:
    - Solutions to these questions must be placed in files **a09q1.rkt**, **a09q2.rkt**, **a09q3.rkt**, respectively, and must be completed in Racket.
    - All solutions must be submitted through MarkUs. Solutions will **not** be accepted through email.
    - Verify your basic test results using MarkUs to ensure that your files were submitted properly and are readable on MarkUs. *Note, however, that passing the basic tests does not guarantee that you will pass all our correctness tests.*
- 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.
- Restrictions:
    - You may only use the built-in functions and special forms introduced in the lecture slides up to and including the module covered by this assignment. A list of these functions can be found on the Assignments web page: https://www.student.cs.uwaterloo.ca/~cs115/#allowed
    - Read each question carefully to see if any additional restrictions apply.
    - Test data for correctness tests will always meet the stated assumptions for consumed values.
- 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.
- (Added November 19, 2019) Unless stated otherwise, a function that consumes a list and produces a list should ensure that the relative order of the elements in the produced list is the same as the relative order in the consumed list.

Language level: Intermediate Student with lambda
Coverage: Module 8

CS115 **Assignment 09** Fall 2019

Due: Wednesday, November 27 @ 10:00 AM (No late submissions)

Useful structures and data definitions:

```
(define-struct auto (brand power fuel))
;; An Auto is a (make-auto Str Num Num)
;; where:
;; brand is the brand/manufacturer of the auto
;; power is the horsepower of the car, measured in hp
;; fuel is the fuel consumption of the car, measured in L/100km
;; requires:
;;   power >=0
;;   fuel >= 0

(define-struct course (subject number title))
;; A Course is a (make-course Sym Nat Str)
;; where:
;; subject is the subject area of the course
;; number is the course number of the course
;; title is the descriptive title describing the content of the
;;    course

(define-struct student (id name age courses))
;; A Student is a (make-student Nat Str Nat (listof Course))
;; where:
;; id is the unique student id for the student
;; name is the name of the student
;; age is the age of the student
;; courses is a list, with one element for each different course
;;   the student is enrolled in
```

CS115 **Assignment 09** Fall 2019

Due: Wednesday, November 27 @ 10:00 AM (No late submissions)

Useful constants for examples:

```
(define car1 (make-auto "Toyota" 45 34.2))
(define car2 (make-auto "Apple" 33 1098))
(define car3 (make-auto "Boo" 100 100))
(define car4 (make-auto "Toyota" 5 4.2))
(define car5 (make-auto "Toyota" 8 2.2))
(define cars (list car1 car2 car3 car4 car5))

(define cs115 (make-course 'CS 115 "Intro to CS"))
(define cs135 (make-course 'CS 135 "Intro to CS"))
(define music115 (make-course 'MUSIC 115 "Pop Music"))
(define math135 (make-course 'MATH 135 "Logic and Proofs"))
(define phys121 (make-course 'PHYS 121 "Mechanics 1"))
(define chem135 (make-course 'CHEM 135 "Explosive Materials II"))

(define stu1 (make-student 213432 "Troy" 22
               (list cs115 math135 phys121)))
(define stu2 (make-student 321413 "Sandy" 23 (list cs115 phys121)))
(define stu3 (make-student 325423 "Lori" 24 (list math135 phys121)))
(define stu4 (make-student 312334 "Valentina" 19
               (list math135 cs135 music115)))
(define stu5 (make-student 100000 "Nobody" 29 empty))
(define stu6 (make-student 200000 "Lefty" 30
               (list chem135 phys121)))
(define students (list stu1 stu2 stu3 stu4 stu5 stu6))
```

Additional testing information:

For testing, it is perfectly acceptable to give two expressions (rather than one expression and one value) to check-expect. By this, we mean that the following is perfectly acceptable for testing:

```
(check-expect (+ 2 6) (expt 2 3))
```

The above will verify that 8 is equal to 8, by evaluating both expressions.

For this and future assignments, some examples given in the problem descriptions use this technique. For instance, in question 1 of the assignment, we point out that:

```
(check-expect (find-auto "Toyota" 5 89.1 cars) (list car1 car4 car5))
```

would be perfectly acceptable. You may use this same technique in testing, in order to decrease the amount of typing.

CS115 **Assignment 09** Fall 2019

Due: Wednesday, November 27 @ 10:00 AM (No late submissions)

Question 1: Find Auto

Write the Racket function `find-auto` that consumes 4 parameters: the desired brand of Auto, the minimum power required, the maximum fuel consumption, and a list of Autos. The function should produce a list of Autos which are the desired brand of Auto, with the minimum power and maximum fuel consumption requirements met. For example:

- `(find-auto "Toyota" 5 5.0 cars) => (list car4 car5)`
- `(find-auto "Apple" 32.6 70 cars) => empty`

Note for this function, you may not use explicit recursion, *named helper functions*, nor `local`. You must use abstract list functions and `lambda`.

Submit your solution in the file **a09q1.rkt**.

Question 2: Best Auto

Not satisfied to find any particular Auto in a list of Autos, you would like to take the best elements of a list of Autos and create a new Auto with those properties.

Write a function `best-auto` which consumes a non-empty list of Autos and produces an Auto with the "best" properties of all Autos in the list. By "best", we mean the function will produce an Auto with:

- the brand of the Auto that occurs last alphabetically. You may assume that all brands have the same capitalization rules (e.g., all brands have the first letter capitalized and all remaining letters in lower-case);
- the maximum power of any Auto in the list of Autos;
- the minimum fuel consumption of any Auto in the list of Autos.

For example:

- `(best-auto cars) => (make-auto "Toyota" 100 2.2)`
- `(best-auto (list car2 car3)) => (make-auto "Boo" 100 100)`

Submit your solution in the file **a09q2.rkt**. You may use explicit recursion, abstract list functions, `local` and/or `lambda` for this question. You may not use `reverse` for this question.

Question 3: Students in courses

Write the Racket function `students-in-course` which consumes a course subject, a course number, and a list of Students, and produces a list containing those Students from the consumed list who are registered in a course with that subject and number. For example:

- `(students-in-course 'CS 115 students) => (list stu1 stu2)`
- `(students-in-course 'MATH 492 students) => empty`

Note for this function, you may not use explicit recursion, *named helper functions*, nor `local`. You must use abstract list functions and `lambda`. Submit your solution in the file **a09q3.rkt**.