CS 115 - Winter 2018. Assignment 6. Due: Wednesday, March 7 at 10:00AM

- **Academic Integrity Links**
  - [https://uwaterloo.ca/academic-integrity/basic-info](https://uwaterloo.ca/academic-integrity/basic-info)
  - [https://uwaterloo.ca/library/get-assignment-and-research-help/academic-integrity/academic-integrity-tutorial](https://uwaterloo.ca/library/get-assignment-and-research-help/academic-integrity/academic-integrity-tutorial)

- 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.

- 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.

- 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 can be found on the Course Website, at [https://www.student.cs.uwaterloo.ca/~cs115/built_in](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 groups on Waterloo LEARN. Choose Connect -> Discussions. Read the guidelines for posting questions. Do NOT post any code as part of your questions.

- 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 description 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 4.
Useful structures and data definitions:

(define-struct course
  (subject number title))
;;  A Course is a
;;  (make-course Sym Nat Str))

(define-struct student
  (student-ID name age courses))
;;  A Student is a
;;  (make-student Nat Str Nat (listof Course))

;;  A Student-Debt (SD) is (list Student Nat)
;;  A Student-Debt-List (SDL) is one of
;;  * empty
;;  * (cons SD SDL)

Useful constants for examples:

(define course1
  (make-course 'CS 115 "An Introduction to Computer Science 1"))
(define course2
  (make-course 'CS 116 "An Introduction to Computer Science 2"))
(define course3
  (make-course 'MATH 135 "Algebra for Honours Math"))
(define course4
  (make-course 'MATH 136 "Linear Algebra 1 for Honours Math"))

(define stud1 (make-student 20493453 "Alice" 19
  (list course1 course3))
(define stud2 (make-student 20563432 "Bob" 20
  (list course2 course3))
(define stud3 (make-student 20544632 "Carl" 20 (list course2))
(define stud4 (make-student 20565453 "David" 21
  (list course1 course4)))

(define stud1-debt (list stud1 15000))
(define stud2-debt (list stud2 25000))
(define stud3-debt (list stud3 5000))
(define stud4-debt (list stud4 25000))
(define stud-list (list stud1 stud2 stud3 stud4))
(define debt-list (list stud1-debt stud2-debt stud3-debt stud4-debt))
1. Complete a Racket function called `ascending-digits?` that consumes a non-empty list of natural numbers `anelon` and produces `true` if the digits of each number are in increasing order and `false` otherwise.

   For example:
   a. `(ascending-digits? (list 2467 14579 5789)) => true`
   b. `(ascending-digits? (list 4)) => true`
   c. `(ascending-digits? (list 1 23 456 7890)) => false`
   d. `(ascending-digits? (list 456 233)) => false`

2. Complete a Racket function called `oldest-student` that consumes a non-empty list of Students `anelos` and produces the name of the oldest student. If there is a tie, produce the student that occurs first in the list.

   For example:
   a. `(oldest-student (list stud1 stud3)) => “Carl”`
   b. `(oldest-student (list stud2 stud3)) => “Bob”`
   c. `(oldest-student stud-list) => “David”`

3. Complete a Racket function called `num-enrolled` that consumes a list of Students `alos`, a symbol `subject`, and a natural number `number`. The function produces the number of students enrolled in the course that matches `subject` and `number`. You may assume a student cannot take the same course more than once.

   For example:
   a. `(num-enrolled (list stud1 stud3) 'MATH 135) => 1`
   b. `(num-enrolled empty 'CS 115) => 0`
   c. `(num-enrolled stud-list 'CS 116) => 2`

4. Complete a Racket function called `most-debt-student` that consumes a non-empty Student-Debt-List `stud-debts` and produces the student with the most debt. If there is a tie, produce the student which appeared first in the list.

   For example:
   a. `(most-debt-student (list stud3-debt stud4-debt)) => stud4`
   b. `(most-debt-student (list stud1-debt)) => stud1`
   c. `(most-debt-student debt-list) => stud2`