Deadline: Wednesday, February 26, 2020 at 10:00am on MarkUs

Language level: Beginning Student with List Abbreviations

Coverage: Module 04

Files to submit: `a05q1.rkt, a05q2.rkt, a05q3.rkt`

Assignment Guidelines

- Solutions for Questions 1-3 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:
  - All solutions must be submitted through MarkUs. Solutions will not be accepted through email.
  - For Questions 1-3, verify your basic test results using MarkUs to ensure that your files were submitted properly and are readable on MarkUs.
    Note: passing the basic tests does not guarantee that you will pass all our correctness tests!
- Download the interface files from the course web page to ensure that all function names are spelled correctly, and that each function has the correct number and order of parameters.
- The language level for this assignment is *Beginning Student with List Abbreviations*, which helps with tests and examples. However, *for this assignment* `list` should not appear in any function bodies.
- **Restrictions:**
  - Unless specifically allowed in the description of the question, you may only use the built-in functions and special forms introduced in the lecture slides in Modules 01-04. For details, see 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.
  - **Reminder:** Do NOT copy/paste text from the Assignment PDF to your definitions window. This can cause errors!
- 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.

Plagiarism: The following applies to all assignments in CS115.

All work in CS 115 is to be done individually. The penalty for plagiarism on assignments (first offense) is a mark of 0 on the affected questions and a 5% reduction of the final grade, consistent with School of Computer Science policy. In addition, a letter detailing the offense is sent to the Associate Dean of Undergraduate Studies, meaning that subsequent offenses will carry more severe penalties, up to suspension or expulsion.

To avoid inadvertently incurring this penalty, you should discuss assignment issues with other students only in a very broad and high-level fashion. Do not take notes during such discussions, and avoid looking at anyone else's code, on screen or on paper. If you find yourself stuck, contact the ISA or instructor for help, instead of getting the solution from someone else. Do not consult other books, library materials, Internet sources, or solutions (yours or other people's) from other courses or other terms.

Read more course policies at: https://www.student.cs.uwaterloo.ca/~cs115/#policies

1. **Above Average**

   Write a Racket function called `strings->bools` that consumes a non-empty list of strings, `los`, and produces a list of Boolean values. A string in `los` produces a corresponding `true` in the produced list if the string's length is greater than the average (mean) length of all strings in `los`, and `false` otherwise.

   For example:

   - `(strings->bools (list "sugar" "CS" "Logic Design" "University" "Waterloo"))`
     `⇒ (list false false true true true)`
   - `(strings->bools (list "abc")) ⇒ (list false)`
   - `(strings->bools (list "a" "aa" "aaa" "aaaa" "aaaaa"))`
     `⇒ (list false false false true true)`

   Place your definitions into the file **`a05q1.rkt`**.

2. **Messy-Alphabetical Order**

   For this problem, we will define "messy words" as **non-empty** strings that:

   - must contain at least one letter, uppercase or lowercase,
   - may contain numbers, and
   - do not contain any other types of characters (no spaces, no special characters).

   We need a way of ordering messy words. For that, we have "messy-alphabetical order". For example, the list `(list word1 word2)` will be considered to be in messy-alphabetical order if, after removing all numbers from `word1` and `word2`, that the resulting words are in alphabetical order. Alphabetical order ignores the case of letters, so it is different from lexicographical order. Empty lists are always considered to be in messy-alphabetical order.

   (**Note:** empty lists are permitted, empty strings are not.)

   Write a Racket predicate `messy-alpha?` that consumes a list of messy words `mess`. The predicate will produce `true` if the strings in `mess` are in "messy-alphabetical" order as described above, and `false` otherwise.

   Sample Example Set 1: comparing normal words. Here, messy-alphabetical order preserves the usual alphabetical order, ignoring the case of letters.

   - `(messy-alpha? (list "ware" "wear" "weir")) ⇒ true`
   - `(messy-alpha? (list "abc" "ABC" "aBC" "AbC")) ⇒ true`
   - `(messy-alpha? (list "DEF" "ab")) ⇒ false`

Sample Example Set 2: comparing messy words.

- `(messy-alpha? (list "9wa123re1" "0w1e2a3r4"))` ⇒ `true`
- `(messy-alpha? (list "ware" "w1e2a3r4"))` ⇒ `true`
- `(messy-alpha? (list "11abc" "A33BC" "aB22C" "1A2b3C4"))` ⇒ `true`
- `(messy-alpha? (list "20DEF123" "21ab123"))` ⇒ `false`

Recall: the built-in function `string<=?` considers the case of letters when comparing strings! For example, `(string<=? "DEF" "ab")` ⇒ `true`, but your function would not consider them to be in messy-alphabetical order! (see last example of Sample Example Set 1)

Note: You are permitted to use the built-in functions `string-ci>=?` and `string-ci<=?`. These functions are part of the permitted string functions from Module 2, in Section 1.11 of the racket documentation.

Place your definitions into the file **`a05q2.rkt`**.

3. **Negative Binary Numbers**

Binary numbers are numbers where the digits are only 0s and 1s, for example 10110. For this assignment we will only be considering binary numbers that start with a 1, for example numbers like 10001 but not numbers like 0001 or 010101. The ones' complement of a binary number switches all of the 0s to 1 and all of the 1s to 0 in the number. It can be used to negate a binary number. For example, the ones' complement of 10110 is 1001, **not** 01001, since we ignore the leading 0.

Write a function called `negate-binary` that consumes a list of natural numbers, `alist`, and produces a list containing the ones complement of the numbers in the list that are valid binary numbers (i.e. numbers that only contain 0s and 1s). The ones' complement numbers in the produced list must be in the same relative order as the original binary numbers appear in `alist`. You may assume none of the numbers have leading 0s; in other words, `alist` will not contain any numbers like `0001`.

For example,

- `(negate-binary (list 1000 101010 123 111))` ⇒ `(list 111 10101 0)`
- `(negate-binary (list 95 137 401))` ⇒ `empty`

Notes:

- The functions `number->string` and `string->number` may be helpful.
- Generally for this question, you do not need to do any extra work to avoid or handle leading 0s. In particular `string->number` produces a number without leading 0s. For example:
  `(string->number "001")` ⇒ `1`

Place your definitions into the file **`a05q3.rkt`**.