Assignment 10
Due Wednesday, August 5 at 10:00 am (no late submissions)

Assignment Guidelines.

- This assignment covers material in Module 10, and throughout the course.
- Submission details:
  - Solutions to these questions must be placed in files `a10q1.rkt`, `a10q2.rkt`, `a10q3.rkt`, and `a10q4.rkt`, respectively, and must be completed using Racket *Intermediate Student with lambda*.
  - Unless otherwise indicated in the question you may use only 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 at `https://www.student.cs.uwaterloo.ca/~cs115/#allowed`
  - 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.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the CS115 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page.
  - For the design recipe, helper functions only require a purpose, a contract and an example.
- Restrictions:
  - Read each question carefully for additional restrictions.

! Do not compute any value more than once. For example, if n is the length of `L`, the following code computes `(length L)` n times:

```
(define (addlen L)
  (local [(define (add-len x) (+ x (length L)))]
    (map add-len L)))
```

Do not do this kind of thing; instead, use `local` constants.

! Do not use `lambda` on this assignment. Use `local` helper functions instead.

! Do not write any non-`local` helper functions on this assignment.
You may use non-`local` functions given in the interface file.

- **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 use `lambda` or any non-`local` helper functions on this assignment! Read the restrictions on the cover page.

1. **Scaling Values.** Sometimes we have a dataset where the exact values do not matter, but only how the values compare to each other. Which values are closest to the smallest or closes to the largest, for example.

Exercise

Write a function `(squash L)` that consumes a `(listof Num)` and returns a list where all values have been shifted and scaled in the same way so that the largest value is 1 and the smallest value is 0.
For example,

```
(check-expect (squash (list 25 13 5)) (list 1 0.4 0))
(check-expect (squash (list 134 100 123 200)) (list 0.34 0 .23 1))
```

Hint

Don't forget to consider if your function requires the data to have certain properties!

2. ***z*-score.** The *z*-score is the number of standard deviations a value is from the mean, given by the formula

$$z = \frac{x - \mu}{\sigma}$$

Where μ is the mean of some population, and σ is the standard deviation of that population.

Exercise

Write a function `(z-scores L)` that consumes a `(listof Num)` and returns a `(listof Num)` representing the *z*-scores of the values in the list.
For example, the mean of `(list 5 5 8 8 8 8 8 11 11)` is 8. The standard deviation is

$$\sqrt{\frac{(5-8)^2+(5-8)^2+(8-8)^2+(8-8)^2+(8-8)^2+(11-8)^2+(11-8)^2}{9}} = \sqrt{\frac{36}{9}} = \sqrt{4} = 2$$

Since the mean is 8 and the standard deviation is 2, the *z*-score of the 5s is -1.5. So
`(z-scores (list 5 5 8 8 8 8 8 11 11)) => (list -1.5 -1.5 0 0 0 0 0 1.5 1.5)`
Further examples:
`(z-scores (list 8 4 8 4)) => (list 1 -1 1 -1)`, since the mean is 6 and the standard deviation is 2.

Hint

Functions to compute mean and standard deviation are included in the interface file. Use them!

You may assume the standard deviation of the values is non-zero (include this as a requirement for your function). That is, it is OK if your function does not work for a list containing only one value, such as `(list 8 8 8 8)`.

3. Mucking with Strings.

Exercise

Write a function `(prefix-smallest-letter L)` that consumes a `(listof Str)`, finds in all the strings the smallest character (according to **`char<?`**), and adds this `Char` at the front of each value in `L`.
For example,

```
(check-expect (prefix-smallest-letter (list "hello" "how" "are" "you"))
              (list "ahello" "ahow" "aare" "ayou"))  ; smallest is #\a
(check-expect (prefix-smallest-letter (list "hello" "how" "is" "you"))
              (list "ehello" "ehow" "eis" "eyou"))   ; smallest is #\e
(check-expect (prefix-smallest-letter (list "I" "am" "Sam"))
              (list "II" "Iam" "ISam"))              ; smallesst is #\I
(check-expect (prefix-smallest-letter (list "I" "like" "char-functions"))
              (list "-I" "-like" "-char-functions")) ; smallest is #\-
```

Hint

Write a [local] helper function that returns the smallest character in a `(listof Char)`. For example,

```
(smallest-char (string->list "hello")) => #\e; since (char<? #\e #\h) => #true, etc.
(smallest-char (string->list "hello&goodbye")) => #\&; since (char<? #\& #\e) => #true
```

4. Padding Strings.

Exercise

Write a function `(pad-to-same L)` that consumes a `(listof Str)`, and “pads” the values by adding as many `"."` as necessary to make all values as long as the longest.
For example,

```
(check-expect (pad-to-same (list "hello" "how" "r" "u?"))
              (list "hello" "how.." "r...." "u?..."))
(check-expect (pad-to-same (list "I" "said" "TANSTAAFL!"))
              (list  "I........." "said......" "TANSTAAFL!"))
```