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.
  - 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 \( \text{length} L \) \( n \) times:

```racket
(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.

- 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.
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 close 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,

(squash (list 25 13 5)) => (list 1 0.4 0)

(squash (list 134 100 123 200)) => (list .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 \( \mu \) is the mean of some population, and \( \sigma \) 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

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

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.

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).

**Hint**

Create two local constants: the mean, and the standard deviation.
3. Local Lists.

Write a function \( \text{add-max-min L} \) that consumes a non-empty \((\text{listof Int})\), and adds the smallest and largest value to each value in \( L \).

For example, in \((\text{list } 2 3 5 7)\), the smallest value is 2, and the largest is 7, so we add \( 2 + 7 = 9 \) to each value, and return \((\text{list } 11 12 14 16)\).

4. Formatting Tabular Data. Often it is useful to display data in a table. Earlier we created a table as a \((\text{listof (listof Num)})\). However, Racket doesn’t know about tabular data, so when its is printed, the columns don’t line up.

   For example, I can define a table of data that is easy to read:

\[
\text{define data (list (list 17 15 6 32768) (list 3 6000 3 2) (list 63 502 16 2344) (list 0 5 10 150))}
\]

   ... but Racket will print it like this:

\[
\text{(list (list 17 15 6 32768) (list 3 6000 3 2) (list 63 502 16 2344) (list 0 5 10 150))}
\]

   I would prefer to have the columns of equal width in all rows. Convert each row to a \text{Str}, like so:

\[
\text{(list " 17 15 6 32768" " 3 6000 3 2" " 63 502 16 2344" " 0 5 10 150")}
\]

Write a function \( \text{format-table T} \) that consumes a \((\text{listof (listof Nat)})\) and returns a \((\text{listof Str})\) formatted in this manner.

   A few notes:
   \- The columns are all the same width
   \- The width of the columns is enough to fit the longest item, with one space between columns.

   Another example:

\[
\text{format-table (list (list 2019 12 11 10 9 8) (list 1 2 4 5 6 7))}
\]

\[
=> \text{(list " 2019 12 11 10 9 8" " 1 2 4 5 6 7")}
\]