CS 135 Fall 2019
Becker, Clarke, Giesbrecht, Holtby, Lanctot, Reetz

Assignment: 3

Due:	Tuesday, October 1st, 9:00 pm
Language level:	Beginning Student
Files to submit:	`pnormp.rkt`, `threestrings.rkt`, `belowspline.rkt`, `shortwalk.rkt`
Warmup exercises:	9.1.1, 9.1.2
Practice exercises:	9.1.3

- **Make sure you read the OFFICIAL A03 post on Piazza** for the answers to frequently asked questions.
- This assignment covers concepts up to Slide 18 of Module 06. Unless otherwise specified, you may only use Racket language features we have covered up to that point. You may also use the builtin functions `second` and `third` if you find them useful. As well, you have not seen recursion up to this point in the course, and it is not necessary nor recommended for this assignment.
- The names of functions we tell you to write, and symbols and strings we specify must match the descriptions in the assignment questions exactly. Any discrepancies in your solutions may lead to a severe loss of correctness marks. Basic tests results will catch many, but not necessarily all of these types of errors.
- Policies from Assignment 2 carry forward.

Here are the assignment questions you need to solve and submit.

1. In this question you will perform step-by-step evaluations of Racket programs, as you did in assignment one. Please review the instructions on stepping in A01.

   To begin, visit this web page:

   `https://www.student.cs.uwaterloo.ca/~cs135/stepping`

   Note: the use of `https` is important; that is, the system will not work if you omit the `s`. This link is also under the Assignments menu on the course web page.

   When you are ready, complete the five required questions under the "Module 6: Lists" category, using the semantics given in class for Beginning Student.

2. Let p be any positive integer. The p-norm of a list of numbers $v = (v_1, v_2, ..., v_n)$ is defined as

$$\|v\|_p = \sqrt[p]{\sum_{i=1}^{n} |v_i|^p}, \quad \text{and its } p\text{th power by} \quad \|v\|_p^p = \sum_{i=1}^{n} |v_i|^p.$$

i.e., $\|v\|_p$ is the pth root of the sum of the pth powers of the absolute values of the entries of v. $\|v\|_p^p$ is similar, but conveniently doesn't involve taking a pth root. In later courses like linear algebra, scientific computation and machine learning, p-norms will be very useful!

For example, with $p = 3$ and $v = (3, -4, 5)$ then

$$\begin{aligned} \|v\|_p &= \|(3,-4,5)\|_3 = \sqrt[3]{|3|^3 + |-4|^3 + |5|^3} = \sqrt[3]{27+64+125} = \sqrt[3]{216} \\ \|v\|_p^p &= \|(3,-4,5)\|_3 = |3|^3 + |-4|^3 + |5|^3 = 27+64+125 = 216 \end{aligned}$$

You are to write a function `pnormp` which consumes a positive integer p and a list v with up to 3 numbers in it, and produces $\|v\|_p^p$. The p-norm of an empty list is 0.

Please put your function into a file called `pnormp.rkt`.

Here are some examples:

```
(check-expect (pnormp 3 (cons 3 (cons -4 (cons 5 empty)))) 216)
(check-expect (pnormp 4 (cons 2 (cons -3 empty))) 97)
(check-expect (pnormp 1 (cons -2 empty)) 2)
```

3. (a) Write a function `in-order?` which consumes three strings S_1, S_2 and S_3, and produces `true` if $S_1 \leq S_2$ and $S_2 \leq S_3$, and `false` otherwise, using lexicographic string comparison (i.e., the `string<=?` function).

   Here are some examples:

   ```
   (check-expect (in-order? "a" "b" "c") true)
   (check-expect (in-order? "a" "a" "c") true)
   (check-expect (in-order? "b" "a" "b") false)
   ```

   (b) Write a function `sort3` which consumes a list of three strings and produces a list of the same three strings in lexicographic order. You can use the `in-order?` function you developed in part (a).

   You may assume the contract for `sort3` is

   ```
   ;; sort3: (listof Str) → (listof Str)
   ;; requires: list is length 3
   ```

   Here are some examples:

   ```
   (check-expect
        (sort3 (cons "a" (cons "b" (cons "c" empty))))
        (cons "a" (cons "b" (cons "c" empty))))
   ```

```
(check-expect
     (sort3 (cons "b" (cons "b" (cons "c" empty))))
     (cons "b" (cons "b" (cons "c" empty))))
(check-expect
     (sort3 (cons "c" (cons "b" (cons "a" empty))))
     (cons "a" (cons "b" (cons "c" empty))))
(check-expect
     (sort3 (cons "a" (cons "b" (cons "a" empty))))
     (cons "a" (cons "a" (cons "b" empty))))
```

Hint: Write down all the possible orders of 3 strings and build your code around this. You may also find the functions `second` and `third` convenient.

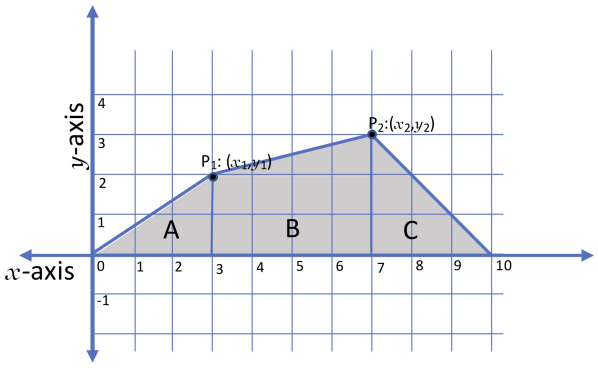
(c) Given a list of three strings, write a function `find-second` which produces the second largest string in the list (using lexicographic ordering). If all three strings are equal, produce `empty`. You may use the `sort3` function you developed in part (b).

Here are some examples:

```
(check-expect (find-second (cons "c" (cons "a" (cons "b" empty))))
   "b")
(check-expect (find-second (cons "c" (cons "c" (cons "a" empty))))
   "a")
(check-expect (find-second (cons "a" (cons "a" (cons "a" empty))))
   empty)
(check-expect (find-second (cons "c" (cons "a" (cons "a" empty))))
   "a")
```

Please put your functions into a file called `threestrings.rkt`.

4. Suppose we are given two points on the plane, $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, where $0 < x_1 < x_2 < 10$ and $y_1 > 0$ and $y_2 > 0$. Drawing a line from $(0,0)$ through P_1, then through P_2, then to $(10,0)$ gives a *spline* (a curve made of straight-line segments) as follows:



Given an additional point (a,b), we want to know if this point lies within the shaded quadrilateral between the spline and the x-axis (i.e., in one of the shaded areas A, B or C). Points on the line are considered to be in the shaded area.

(a) Points on the plane will be represented by a list of two numbers. For example `(cons`

`3 (cons 5 empty))` represents $(3,5)$. To assist in creating points, write a function `make-point` which consumes two numbers x and y, and produces the list of length 2 containing x and y. To assist in accessing the coordinates of points, write a function `x-coord` which consumes a point and produces its x coordinate. Similarly, write a function `y-coord`, which consumes a point and produces its y coordinate.

Here are some examples:

```
(define Q (make-point 3 2))
(check-expect (x-coord Q) 3)
(check-expect (y-coord Q) 2)
```

(b) Write a function `below-spline?` which consumes 3 points (x_1,y_1), (x_2,y_2), (a,b), represented as in part (a), and produces `true` if (a,b) lies between the curve and the x-axis (i.e., the grey area in the diagram), and `false` otherwise. Points on the lines are considered to be between the curve and the axis. You should use the functions `make-point`, `x-coord` and `y-coord` as appropriate.

Here are some examples:

```
(define P1 (make-point 3 2))
(define P2 (make-point 7 3))
(check-expect (below-spline? P1 P2 (make-point -1 2)) false)
(check-expect (below-spline? P1 P2 (make-point 1 1)) false)
(check-expect (below-spline? P1 P2 (make-point 1 1/2)) true)
(check-expect (below-spline? P1 P2 (make-point 3 2)) true)
(check-expect (below-spline? P1 P2 (make-point 5 (+ 5/2 1/100)))
   false)
```

Please put your functions into a file called `belowspline.rkt`.

5. A *walk* on a plane consists of list of directions (i.e. north, south, east, or west), and a list of distances (positive numbers) which we follow one after another as steps.

   For example, suppose we start at a point $(3,4)$, with a list of directions (north, west), and a list of distances $(2,5)$. Our first step would be to go north a distance of 2. Our second step would be to go west a distance of 5. This would leave us at coordinate $(-2,6)$ at the end of our walk. Note that we regard the x-axis as running from west to east, and the y-axis as running from south to north.

   Points on the plane are represented in Racket as in Q4 by a list of two numbers. You can use the functions `make-point`, `x-coord` and `y-coord` that you created for Q4 by pasting these functions and their definitions into your solution file.

   Please put your functions into a file called `shortwalk.rkt`.

   (a) Write a function `make-step` which consumes a starting point on the plane (a list of two numbers), a direction (a symbol, either `'N`, `'S`, `'E`, or `'W`) and a distance (a positive number) and produces a new point (a list of two numbers) which is the position after moving the specified distance in the specified direction.

Here are some examples:

```
(check-expect (make-step (make-point 2 3) 'E 4) (make-point 6 3))
(check-expect (make-step (make-point 5 -3) 'S 2) (make-point 5 -5))
```

(b) Write a function `two-steps` which consumes a starting point P, a list of two directions (D_1, D_2) , and a list of two distances (d_1, d_2), and produces a new point on the plane which is the position after following the two steps. That is, starting at P, move d_1 steps in direction D_1, then d_2 steps in direction D_2.

Here are some examples:

```
(check-expect
     (two-steps (make-point 1 -1)
         (cons 'N (cons 'E empty))
         (cons 2 (cons 3 empty)))
     (make-point 4 1))

(check-expect
    (two-steps (make-point 3 3)
         (cons 'E (cons 'S empty))
         (cons 4 (cons 1 empty)))
    (make-point 7 2))
```