What is wrong with each of the following?

- (* (5) 3)
- (+ (* 2 4)
- (5 * 14)
- (* + 3 5 2)
- (/ 25 0)

Exercise

Write (discard-bad L lo hi). It consumes a (listof Num) and two Num. It returns the list of all values in L that are between lo and hi, inclusive.

(discard-bad (list 12 5 20 2 10 22) 10 20) => (list 12 20 10)

Exercise

Write a function (distances xs ys) that consumes two lists: the first contains x values, and the second contains y values. The output is a list containing the distance of each point (x, y) from (0, 0).

(distances (list 3 0 2) (list 4 7 2)) => (list 5 7 #i2.828427)

(Since (3, 4) is at distance 5; (0, 7) is at distance 7; and (2, 2) is at distance \( \sqrt{8} \approx 2.828427 \).)

Exercise

Write a function prod that returns the product of a (listof Num).

(prod (list 2 2 3 5)) => 60

Exercise

Read the documentation on string-length. Write a function that returns the total length of all the values in a (listof Str).

(total-length (list "hello" "how" "r" "u?")) => 11

Exercise

Write a Racket function corresponding to

\[ g(x, y) = x\sqrt{x} + y^2 \]

((sqr n) computes \( n^2 \) and (sqr n) computes \( n^2 \).)

Exercise

Use filter to write a function that consumes a (listof Num) and keeps only values between 10 and 30, inclusive.

(keep-inrange (list -5 10.1 12 7 30 3 19 6.5 42)) => (list 10.1 12 30 19)

Exercise

Write a function (countdown-by top step) that returns a list of Nat so the first is top, the next is step less, and so on, until the next one would be zero or less.

(countdown-by 15 3) => (list 15 12 9 6 3)

(countdown-by 14 3) => (list 14 11 8 5 2)

Exercise

The factorial function, \( n! \), returns the product of the numbers from 1 to \( n \). For example, \( 4! = 1 \times 2 \times 3 \times 4 = 24 \).

Write a function (factorial n) that returns \( n! \).

(factorial 5) => 120

(factorial 1) => 1

Exercise

Change ponder so muck-after-str also removes every value that immediately follows the word “POP”.

E.g. (muck-after-str (list 5 7 "POP" 4 3)) => (list 5 7 3)

Exercise

Trace the program:

(+ (remainder (- 10 2) (quotient 10 3)) (* 2 3))
<table>
<thead>
<tr>
<th>Exercise</th>
<th>Write a recursive function <code>keep-evens</code> that consumes a <code>(listof Int)</code> and returns the list of even values. That is, use recursion to duplicate the following function:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(define (keep-evens L) (filter even? L))</td>
</tr>
<tr>
<td>Exercise</td>
<td>Using <code>lambda</code> and <code>map</code>, but no [named] helper functions, write a function that consumes a <code>(listof Num)</code> and returns a list containing the cube of each <code>Num</code>.  $(x^3)$</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Exercise | Using `foldr`, write a function `(keep-multiples n L)` that returns the list containing all the values in `L` that are multiples of `n`. That is, it acts like `(filter (lambda (x) (= 0 (remainder x n))) L)`.  
`(keep-multiples 3 (list 1 2 3 4 5 6 7)) => (list 3 6)`                                                                                                                                                                                                         |
| Exercise | Perform a trace of  
`(or (< 7 4) (= 3 3) (> 7 4) (> 0 (/ 3 0)))`                                                                                                                                                                                                                                                                               |
|          |                                                                                                                                                                                                                                                                                                                                 |
| Exercise | Change `ponder` so `muck-after-str` also changes every value that immediately follows the word “SQUARE” be the square of that number.  
E.g. `(muck-after-str (list 5 7 "SQUARE" 4 3)) => (list 5 7 16 3)`                                                                                                                                                                                                 |
| Exercise | Write a function `drop-e` that converts a `Str` to a `(listof Char)`, replaces each `\e` with a `\*`, and converts it back to a `Str`.  
`(drop-e "hello world, how are you?") => "h*llo world, how ar* you?"`                                                                                                                                                                                                 |
| Exercise | Use `foldr` to write a function `(add-n-each n L)` that adds `n` to each value in `L`.  
`(add-n-each 7 (list 2 4 8)) => (list 9 11 15)`                                                                                                                                                                                                                                                                       |
| Exercise | Write a function `remove-second` that consumes a list of length at least two, and returns the same list with the second item removed.  
`(remove-second (list 2 4 6 0 1)) => (list 2 6 0 1)`                                                                                                                                                                                                                                                                           |
| Exercise | Read about stacks, and be amazed.                                                                                                                                                                                                                                                                                           |
| Exercise | Write a recursive function `list-max` that consumes a nonempty `(listof Int)` and returns the largest value in the list.                                                                                                                                                                                                                                                                   |
| Exercise | Complete `join-names`.  
`; (join-names G S) Make a list of full names from G and S.  
`; join-names: (listof Str) (listof Str) -> (listof Str)  
`; Example:  
`(check-expect (join-names gnames snames)  
  (list "Joseph Hagey" "Burt Matthews" "Douglas Wright"  
    "James Downey" "David Johnston"))`                                                                                                                                                                                                                                  |
| Exercise | Write a function `(add-total L)` that consumes a `(listof Num)`, and adds the total of the values in `L` to each value in `L`.  
`(add-total (list 2 3 5 10)) => (list 22 23 25 30)`                                                                                                                                                                                                                                                                       |
Exercise
Write a function (multiply-each L n). It consumes a (listof Num) and a Num, and returns the list containing all the values in L, each multiplied by n.

(multiply-each (list 2 3 5) 4) => (list 8 12 20)

Exercise
Write a function that consumes a (listof Num) and returns the list containing just the values which are greater than or equal to the average (mean) value in the list.

Exercise
Write a function (absdiff a b) that consumes two (listof Int) and returns a (listof Nat) containing the absolute value of the difference between corresponding values.

(absdiff (list 1 3 5 7) (list 7 3 6 1)) => (list 6 0 1 6)

Exercise
Perform a trace of
(and (= 3 3) (> 7 4) (< 7 4) (> 0 (/ 3 0)))

Exercise
Make the word "ADD" add up the two values that come after it.
(muck-after-str (list 5 7 "ADD" 7 3 5)) => (list 5 7 10 5)

Exercise
Complete countdown using recursion. (Hint: use cons.)

;;; (countdown n) return a list of the natural numbers from n down to 0.
;;; countdown: Nat -> (listof Nat)
;;; Examples:
;;; (check-expect (countdown 3) (cons 3 (cons 2 (cons 1 (cons 0 '())))))
;;; (check-expect (countdown 5) (list 5 4 3 2 1 0))

Exercise
Write (squash-bad lo hi L). It consumes two Num and a (listof Num). Values in L that are greater that hi become hi; less that lo become lo.

(squash-bad 10 20 (list 12 5 20 2 10 22)) => (list 12 10 20 10 10 20)

Exercise
Write a function count-at that consumes a Str and counts the number of times \a or \t appear in it.

(count-at("A cat sat on a mat") => 7)

Exercise
Using lambda and filter but no [named] helper functions, write a function that consumes a (listof Str) and returns a list containing all the strings that have a length of 4.

(keep4 (list "There's" "no" "fate" "but" "what" "we" "make" "for" "ourselves") => (list "fate" "what" "make")

Exercise
Write a recursive function (sum-between n b) than consumes two Nat, with n ≥ b, and returns the sum of all Nat between b and n.

(sum-between 5 3) => (+ 5 4 3) => 12

Exercise
Write a function that returns the average (mean) of a non-empty (listof Num).

(average (list 2 4 9)) => 5
(average (list 4 5 6 6)) => 5.25
Recall that (length L) returns the number of values in L.

Exercise
Use foldr to write a function that behaves like map.

(my-map sqr (list 4 5 3)) => (list 16 25 9)
Exercise: Use `foldr` to write a function that behaves like `filter`.

```
(my-filter odd? (list 4 5 9 6)) => (list 5 9)
```

Exercise: Write a recursive function `sum` that consumes a `(listof Int)` and returns the sum of all the values in the list.

```
(sum (list 6 7 42)) => 55
```

That is, use recursion to duplicate the following function:

```
(define (sum L) (foldr + 0 L))
```

Exercise: Write a function `(times-row n len)` that returns the nth row of the times table. This should be a list of length len. Write you function in the form `(map ... (range 1 (+ len 1) 1))`.

```
(times-row 3 4) => (list 3 6 9 12)
(times-row 5 3) => (list 5 10 15)
```

Exercise: Write a function `(times-table len)` that returns the n×n times table. Use `times-row` as a helper function.

```
(timestable 5) =>
(list (list 1 2 3 4 5)
         (list 2 4 6 8 10)
         (list 3 6 9 12 15)
         (list 4 8 12 16 20)
         (list 5 10 15 20 25))
```

Exercise:
```
(define y 3)
(define (g x) (+ x y))
(g 5)
```

Exercise:
```
(define x 4)
(define (f x) (* x x))
(f 3)
```

Exercise: Complete `enumerate-words`.

```
;; (enumerate-words l) format the values in l with their index, like:
;; 1. first item
;; 2. second item
;; 3. third item
;; enumerate-words: (listof Str) -> (listof Str)
;; Examples:
(check-expect (enumerate-words (list "Mercury" "Venus" "Earth" "Mars"
                                "Jupiter" "Saturn" "Uranus" "Neptune"))
              (list "1. Mercury" "2. Venus" "3. Earth" "4. Mars"
```

Exercise: Complete `countdown-to` using recursion.

```
;; (countdown-to n b) return a list of Int from n down to b.
;; countdown-to: Int Int -> (listof Int)
;; Examples:
(check-expect (countdown-to 2 0) (cons 2 (cons 1 (cons 0 '()))))
(check-expect (countdown-to 5 2) (list 5 4 3 2))
```
Exercise:  
Experiment with fold-sub. Describe how it behaves, and why.

\[
\text{(define (fold-sub \ L) (foldr - 0 \ L)) (fold-sub (list 6 5 2)) => ?}
\]

Exercise:  
Trace the program: \((\sqrt{n})\) computes \(\sqrt{n}\) and \((\text{sqr} \ n)\) computes \(n^2\)

\[
\text{(define (disc \ a \ b \ c) (sqrt (- (sqr \ b) (* 4 (* a c)))))}
\]
\[
\text{(define (proot \ a \ b \ c) (/ (+ (- 0 b) (disc \ a \ b \ c)) (* 2 a)))}
\]
\[
\text{(proot 1 3 2)}
\]

Exercise:  
Digital signals are often recorded as values between 0 and 255, but we often prefer to work with numbers between 0 and 1.

Write a function \((\text{squash-range} \ \text{L})\) that consumes a \((\text{listof Nat})\), and returns a \((\text{listof Num})\) so numbers on the interval \([0, 255]\) are scaled to the interval \([0, 1]\).

\[
\text{(squash-range (list 0 204 255)) => (list 0 0.8 1)}
\]

Exercise:  
Write a function that consumes a \((\text{listof Num})\) and returns a list with each number doubled.

The following function works. Rewrite it using \(\text{foldr}\), without using \(\text{map}\).

\[
\text{(define (double \ n) (* \ n 2))}
\]
\[
\text{(define (double-each \ L) (map double \ L))}
\]

Exercise:  
Use \(\text{filter}\) to write a function that keeps all items which are a \((\text{list} \ a \ b \ c)\) containing a Pythagorean triple \(a < b < c: a^2 + b^2 = c^2\)

\[
\text{(check-expect (pythagoreans (list \ 1 \ 2 \ 3) (list \ 3 \ 4 \ 5) (list \ 5 \ 12 \ 13) (list \ 4 \ 5 \ 6)))}
\]
\[
\text{(list (list \ 3 \ 4 \ 5) (list \ 5 \ 12 \ 13)))}
\]

Exercise:  
Given that \(\text{use-foldr}\) consumes a \((\text{listof Nat})\):

\[
\text{(define (use-foldr \ L) (foldr myfun "some-str" \ L))}
\]

1. What is the contract for \(\text{myfun}\) ?  
2. What is the contract for \(\text{use-foldr}\) ?

Exercise:  
Write a function that consumes a \(\text{Num}\), and returns

- "big" if \(80 < x \leq 100\),
- "small" if \(0 < x \leq 80\),
- "invalid" otherwise.

Exercise:  
Complete the function \((\text{admission after5? age})\) that returns the admission cost.

\[
; ; \text{admission: Bool Nat -> Num}
\]
\[
\text{(check-expect (admission \ #true \ 6) 0)}
\]
\[
\text{(check-expect (admission \ #false \ 6) 5)}
\]
\[
\text{(check-expect (admission \ #false \ 32) 10)}
\]

Exercise:  
Complete the function \((\text{list-cubes})\).

\[
; ; (\text{list-cubes} \ n) \text{ return the list of cubes from } 1^3 \text{ to } n^3 \text{.}
\]
\[
; ; \text{list-cubes: Nat -> (listof Nat)}
\]
\[
; ; \text{Examples:}
\]
\[
\text{(check-expect (list-cubes 4) (list 1 8 27 64))}
\]
Exercise Write a function `acronymize` that consumes a `(listof Str)`, where each `Str` is of length at least 1, and returns a `Str` containing the first letter of each item in the list.

```
(acronymize (list "Portable" "Network" "Graphics")) => "PNG"
(acronymize (list "GNU" s" "Not" "UNIX")) => "GNU"
```

Exercise Write a function `(collatz-next sk)` that consumes a `Nat` representing an item in a Collatz sequence, and returns the next item in the sequence.

```
(collatz-next 3) => 10
(collatz-next 12) => 6
```

Exercise Using `lambda` just once and `foldr` just once, and no [named] helper functions, write a function that consumes a `(listof Int)` and returns the sum of all the even values.

```
(sum-evens (list 2 3 4 5)) => 6
```

Exercise Write a function that consumes a `(listof Str)`, where each `Str` is a person’s name, and returns a list containing a greeting for each person.

```
(greet-each (list "Ali" "Carlos" "Sai")) => (list "Hi Ali!" "Hi Carlos!" "Hi Sai!")
```

Exercise Write a recursive function `(step-sqr-sum-between lo hi step)`, that returns the sum of squares of the numbers starting at `lo` and ending before `hi`, spaced by `step`. That is, duplicate the following function:

```
(define (step-sqr-sum-between lo hi step)
  (foldr + 0 (map sqr (range lo hi step))))
```

Exercise Write a function `(sum-to n)` that consumes a `Nat` and returns the sum of all `Nat` between 0 and `n`.

```
(sum-to 4) => (+ 4 3 2 1 0) => 10
```

Exercise Write a function `times-square` that consumes a `(listof Nat)` and returns the product of all the perfect squares (1, 4, 9, 16, 25, ... ) in the list.

```
(times-square (list 1 25 5 4 1 7)) => (* 1 25 4 1) => 100
```

Exercise Using `cond` and `map`, write a function `neg-odd` that consumes a `(listof Nat)`. The function returns a `(listof Int)` where all odd numbers are negative, and all even numbers positive.

```
(neg-odd (list 2 5 8 11 14 17)) => (list -2 -5 -8 -11 -14 -17)
```

Exercise Write a full design recipe for a function `distance` which computes the distance between (0, 0) and a given point (x, y).

Include `purpose`, `contract`, `examples`, `implementation`, and `tests`.

Exercise Write the helper function `(ponder new-item answer)` that allows muck-after-str to work.

```
(muck-after-str (list 2 7 "X" 3 5)) => (list 2 7 6 5)
```

Exercise

```
(define z 3)
(define (h z) (+ z z))
(h 7)
```
Using `foldr`, write a function `(keep-evens L)` that returns the list containing all the even values in `L`. That is, it acts like `(filter even? L)`.

```lisp
(keep-evens (list 1 2 3 4 5 6)) => (list 2 4 6)
```

Write a function that returns the number of odd numbers in a `(listof Nat)`.
Hint: read the documentation on `remainder`.
Can you do this using `map` and `foldr`? Just using `foldr`?

Write purpose, contract, examples, and tests for:
(1) The absolute value function

Use `define` to create a function `(add-twice a b)` that returns `a + 2b`.

```lisp
(add-twice 3 5) => 13
```

Write a function `(sum-square-difference n)` that consumes a `Nat` and returns the difference between the square of the sum of numbers from 0 to `n`, and the sum of the squares of those numbers.

```lisp
(sum-square-difference 3) => (- (sqr (+ 0 1 2 3)) (+ 0 1 4 9)) => 22
```

Use recursion to complete the function `list-cubes`.

```lisp
;; (list-cubes b t) return the list of cubes from b*b*b to t*t*t.
;; list-cubes: Nat Nat -> (listof Nat)
;; Examples:
(check-expect (list-cubes 2 5) (list 8 27 64 125))
```

Write a function `myfun` that allows `use-foldr` to do something.

Given these definitions:

```lisp
(define foo 4)
(define (bar a b) (+ a a b))
```

What is the value of this expression?

```lisp
(* foo (bar 5 (/ 8 foo)))
```