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

```
(define add-twice (lambda (a b) (+ a (* 2 b))))
(add-twice 3 5)  =>  13
```

Distance
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 3 2) (list 4 7 2))  =>  (list 5 7 #i2.828427)
```

Exercise
```
(define z 3)
(define h (lambda (z) (+ z z)))
(h 7)
```

Exercise
Use **foldr** to write a function `(add-1-each L)` that adds 1 to each value in `L`.

```
(define add-1-each (lambda (L) (foldr (lambda (x lst) (cons (+ x 1) lst)) nil L)))
(add-1-each (list 2 4 8))  =>  (list 3 5 9)
```

Exercise
Complete the function `list-cubes`.

```
;; (list-cubes n) return the list of cubes from 1*1*1 to n*n*n.
;; list-cubes: Nat -> (listof Nat)
;; Examples:
;; (check-expect (list-cubes 4) (list 1 8 27 64))
```

Exercise
```
(define g (lambda (x y) (+ (sqrt x) (* y y))))
(g 4 3)
```

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

Exercise
```
(or (< 7 4) (= 3 3) (> 7 4) (> 0 (/ 3 0)))
```

Exercise
What is wrong with each of the following?
- `(* (5) 3)`
- `(+ (* 2 4)`
- `(5 * 14)`
- `(* + 3 5 2)`
- `(/ 25 0)`

Exercise
```
(define foo 4)
(define bar (lambda (a b) (+ a a b)))
```

What is the value of this expression?
```
(* foo (bar 5 (/ 8 foo)))
```
Exercise: Use `foldr` to write a function that behaves like `filter`.

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

Exercise: Read the documentation on `string-length`.

Write a function that returns the total length of all the values in a `(listof Str)`.

Exercise: Perform a trace of

```scheme
(and (= 3 3) (> 7 4) (< 7 4) (> 0 (/ 3 0)))
```

Exercise: Use `foldr` to write a function `(add-n-each n L)` that adds `n` to each value in `L`.

```scheme
(add-n-each 7 (list 2 4 8)) => (list 9 11 15)
```

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`.

```scheme
(add-total (list 2 3 5 10)) => (list 22 23 25 30)
```

Exercise: Complete `join-names`.

```scheme
;; (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 "David Johnston" "James Downey" "Douglas Wright"
        "Burt Matthews" "Joseph Hagey"))
```

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

```scheme
(check-expect (pythagoreans
  (list (list 1 2 3) (list 3 4 5) (list 5 12 13) (list 4 5 6)))
  (list (list 3 4 5) (list 5 12 13)))
```

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`.

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

Exercise: Write purpose, contract, examples, and tests for:

1. The absolute value function `\|2-\`.
2. A function which computes the GCD of two natural numbers `\|3-\`.
3. Assume a `Graph` is already defined, a function which counts the vertices in a `Graph`.

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

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

Trace the program:

```
(+ (remainder (- 10 2) (quotient 10 3)) (* 2 3))
```

Exercise

Write a function `(double-add a b)` that consumes two `Vector` and returns twice the vector sum of them.

```
(double-add (list 2 3 3) (list 7 4 1)) => (list 18 14 8)
```

Exercise

Write a function `(discard-bad L lo hi)` that 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

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

```
(squash-range (list 0 204 255)) => (list 0 0.8 1)
```

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

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`?

Exercise

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

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

Exercise

Complete the function `(admission after5? age)` that returns the admission cost.

```
;; admission: Bool Nat -> Num
```

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?"
```
Using `cond` and `map`, write a function that consumes a `(list of Int)`. The function makes all odd numbers negative, and all even numbers positive.

\[(\text{neg-odd } (\text{list } 2 5 8 -11 -14 -17)) \Rightarrow (\text{list } 2 -5 8 -11 14 -17)\]

Exercise

Experiment with `fold-sub`. Describe how it behaves, and why.

\[(\text{define } (\text{fold-sub } L) (\text{foldr } - 0 L))\]

\[(\text{fold-sub } (\text{list } 6 5 2)) \Rightarrow ?\]

Exercise

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

\[(\text{define } (\text{disc } a \ b \ c) (\text{sqrt } (- (\text{sqr } b) (* 4 (* a c))))))\]

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

\[(\text{proot } 1 \ 3 \ 2)\]

Exercise

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

\[(\text{my-map } \text{add1 } (\text{list } 6 8 48)) \Rightarrow (\text{list } 7 9 49)\]

Exercise

Write a function `times-square` that consumes a `(list of Nat)` and returns the product of all the perfect squares in the list.

Hint: use `integer?` to check if a value is an integer.

\[(\text{times-square } (\text{list } 1 36 5 4 1 7)) \Rightarrow (* 1 36 4 1) \Rightarrow 144\]

Exercise

Write a function \((\text{multiply-each } L \ n)\). It consumes a `(list of Num)` and a `Num`, and returns the list containing all the values in \(L\), each multiplied by \(n\).

\[(\text{multiply-each } (\text{list } 2 3 5) 4) \Rightarrow (\text{list } 8 12 20)\]

Exercise

\[(\text{define } y 3)\]

\[(\text{define } (g \ x) (+ \ x \ y))\]

\[(g \ 5)\]

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.

\[(\text{remove-second } (\text{list } 2 4 6 0 1)) \Rightarrow (\text{list } 2 6 0 1)\]

Exercise

Write a function `flatten` that consumes a `(list of (list of Any))` and returns a list containing all the values in the lists.

\[(\text{flatten } (\text{list } (\text{list } 1 2) (\text{list } 3 4) (\text{list } 7))) \Rightarrow (\text{list } 1 2 3 4 7)\]

Hint: read the documentation on `append`.

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`. 
Write a function that returns the average (mean) of a (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.

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.

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

Write a function that consumes a (listof Num) and returns the list with each number doubled.

The following function works. Rewrite it using foldr, without using map.

(define (double-each L) (map (lambda (x) (+ x x))) L)

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"

Using lambda and map, but no helper functions, write a function that consumes a (listof Num) and returns a list containing the cube of each Num. \((x^3)\)

Write a function that consumes a Num, and returns
- "big" if \(80 < x \leq 100\),
- "small" if \(0 < x \leq 80\),
- "invalid" otherwise.

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).

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

Given that use-foldr consumes a (listof Nat):

(define (use-foldr L) (foldr myfun "some-str" L))

(1) What is the contract for myfun?
(2) What is the contract for use-foldr?

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

(define (huh? huh?) (+ huh? 2))
(huh? 4)

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.