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

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

Write a function \(\text{flatten}\) that consumes a \((\text{listof (listof Any)})\) 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 \(\text{append}\).

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

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

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

Use recursion to write a function that duplicates the following function:

\[
(\text{def } (f L) (\text{map } (\lambda (x) (+ (sqr x) x)) L))
\]

\[
(\text{define } y 3)
(\text{define } (g x) (+ x y))
(\text{g } 5)
\]

Complete \(\text{join-names}\).

\[
; ; \text{(join-names G S)} \text{ Make a list of full names from G and S.}
; ; \text{join-names: (listof Str) (listof Str) -> (listof Str)}
; ; \text{Example:}
; ; (check-expect \text{(join-names gnames snames)}
; \text{(list } "David Johnston" "James Downey" "Douglas Wright"
; "Burt Matthews" "Joseph Hagey")
\]

Complete \(\text{dict-add}\).

\[
; ; \text{(dict-add d k v) return a new dictionary containing all values in d}
; ; \text{and new value (make-asc k v). Keep data sorted by key.}
; ; \text{If key is already in d, replace its value.}
; ; \text{dict-add: Dict Nat Any -> Dict}
; ; \text{Example:}
; ; (check-expect \text{(dict-add student-dict}
; 7587 (\text{make-student } "George W Bush" "business")
; (\text{list } (\text{make-asc } 6938 \text{ make-student } "Al Gore" "government")
; (\text{make-asc } 7334 \text{ make-student } "Bill Gates" "appliedmath")
; (\text{make-asc } 7587 \text{ make-student } "George W Bush" "business")
; (\text{make-asc } 8838 \text{ make-student } "Barack Obama" "law")))
\]

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

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

```
(times-square (list 1 36 5 4 1 7)) => (* 1 36 4 1) => 144
```

Exercise
Write a recursive 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 `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
```
(define (huh? huh?) (+ huh? 2))
(huh? 4)
```

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`

```
(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
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
Use recursion to complete `append-lists`.

```
;; (append-lists L1 L2) form a list of the items in L1 then L2, in order.
;; append-lists: (listof Any) (listof Any) -> (listof Any)
;; Example:
(check-expect (append-lists (list 3 4) (list 6 8)) (list 3 4 6 8))
```

Exercise
Complete `count-leaves`.

```
;; (count-leaves expr) return the number of leaves in expr.
;; count-leaves: AExp -> Nat
;; Examples:
(check-expect (count-leaves
  (make-ainode '+ (list 2 3 (make-ainode '*' (list 6 7 42)))))) 5)
```
Exercise
Write a function that returns the average (mean) of a \((\text{listof } \text{Num})\).
\[
\text{average (list 2 4 9)} \Rightarrow 5
\]
\[
\text{average (list 4 5 6 6)} \Rightarrow 5.25
\]
Recall that \((\text{length } L)\) returns the number of values in \(L\).

\[
\text{(define } z 3) \\
\text{(define } (h z) (+ z z)) \\
(h 7)
\]

Exercise
Complete expand-bst.
\[
;\; (\text{expand-bst } L \text{ tree}) \text{ add all items in } L \text{ to tree, adding the last first.} \\
;\; \text{expand-bst: (listof Association) BST } \rightarrow \text{ BST} \\
;\; \text{Example:} \\
;\; \text{(check-expect} \\
;\; \text{(expand-bst (list (make-association 4 "four") ) ) } '() \\
;\; \text{make-node 4 "four" '() '()}) \\
;\; \text{(check-expect} \\
;\; \text{(expand-bst (list (make-association 2 "two")} \\
;\; \text{make-association 6 "six")} \\
;\; \text{make-association 4 "four") ) } '() \\
;\; \text{make-node 4 "four"} \\
;\; \text{make-node 2 "two" '() '()}) \\
;\; \text{(make-node 6 "six" '() '()})
\]

Exercise
Complete the function list-cubes.
\[
;\; (\text{list-cubes } n) \text{ return the list of cubes from } 1^*1 \text{ to } n*n*n. \\
;\; \text{list-cubes: Nat } \rightarrow \text{ (listof Nat)} \\
;\; \text{Examples:} \\
;\; \text{(check-expect (list-cubes 4) (list 1 8 27 64))}
\]

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

Exercise
Use recursion to complete the function list-cubes.
\[
;\; (\text{list-cubes } b \ t) \text{ return the list of cubes from } b*b*b \text{ to } t*t*t. \\
;\; \text{list-cubes: Nat Nat } \rightarrow \text{ (listof Nat)} \\
;\; \text{Examples:} \\
;\; \text{(check-expect (list-cubes 2 5) (list 8 27 64 125))}
\]

Exercise
Write a function acronymize that consumes a \((\text{listof } \text{Str})\), where each \(\text{Str}\) is of length at least 1, and returns a \(\text{Str}\) containing the first letter of each item in the list.
\[
\text{(acronymize (list "Portable" "Network" "Graphics")) } \Rightarrow \text{"PNG"}
\]
\[
\text{(acronymize (list "GNU"'s" "Not" "UNIX")) } \Rightarrow \text{"GNU"}
\]
Exercise
Write a recursive function `keep-evens` that consumes a `(listof Int)` and returns the list of even values.
That is, use recursion to duplicate the following function:
```
(define (keep-evens L) (filter even? L))
```

Exercise
Write a function `vector2D+` that consumes two `Posn` and does vector addition. (That is, the new x is the sum of the x values, and the new y is the sum of the y values.)
```
;; (vector2D+ v1 v2) return the vector sum of v1 and v2.
;; vector2D+: Posn Posn -> Posn
;; Example:
(check-expect (vector2D+ (make-posn 2 3) (make-posn 5 8)) (make-posn 7 11))
```

Exercise
Write a function `(find-ldict key dict)` that consumes a `Nat` and a `LDict`. The function returns the value in `dict` associated with the `key`. You may assume `key` appears exactly once in `dict`.
```
(check-expect (find-ldict 6938 student-dict) (list "Al Gore" "government"))
```
You could solve this using recursion, but it might be easier using `filter`.

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
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 recursive function that duplicates the following function:
```
(define (g L) (filter (lambda (x) (= 0 (remainder x 3))) L))
```

Exercise
Complete `keep-close-to-end`. (Hint: use `length`.)
```
;; (keep-close-to-end L) return the items in L that do not exceed their distance from the end.
;; keep-close-to-end: (listof Nat) -> (listof Nat)
;; Example:
(check-expect (keep-close-to-end (list 3 2 1 0)) (list 3 2 1 0))
(check-expect (keep-close-to-end (list 6 3 2 5 4)) (list 3 2))
```

Exercise
Complete `count-leaves`.
```
;; (count-leaves tree) return the number of leaves in tree.
;; count-leaves: SSTree -> Nat
;; Example:
(check-expect (count-leaves tree12) 2)
```
Write a function `prod` that returns the product of a `(listof Num)`.

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

Exercise

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

`(define (fold-sub L) (foldr - 0 L))
(fold-sub (list 6 5 2)) => ?`

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

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

Exercise

Write purpose, contract, examples, and tests for:

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

Exercise

Write a function `(rotate-polar p angle)` that consumes a `Polarcoord` and a `Num` and returns the `Polarcoord` that results from rotating `p` by `angle`.

`;; (rotate-polar p angle) return p rotated by angle.
;; rotate-polar: Polarcoord Num -> Polarcoord
;; Example:
(check-expect (rotate-polar (make-polarcoord 3 0.4) 0.2)
(make-polarcoord 3 0.6))

Exercise

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

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

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

Complete `list=?`

`;; (list=? a b) return true iff a and b are equal.
;; list=?: (listof Any) (listof Any) -> Bool
;; Examples:
(check-expect (list=? (list 6 7 42) (list 6 7 42)) true)
Exercise
Use define to create a function \( \text{add-twice} \ a \ b \) that returns \( a + 2b \).
\( \text{add-twice} \ 3 \ 5 \) \( \Rightarrow \) 13

Exercise
Complete tree-sum.

```scheme
;; (tree-sum tree) return the sum of all keys in tree.
;; tree-sum: SSTree -> Num
;; Example:
(check-expect (tree-sum tree12) 48)
```

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 \( \text{squash-range L} \) that consumes a \( \text{(listof Nat)} \), and returns a \( \text{(listof Num)} \) so numbers on the interval \([0, 255]\) are scaled to the interval \([0, 1]\).
\( \text{squash-range} \ (\text{list} \ 0 \ 204 \ 255) \) \( \Rightarrow \ (\text{list} \ 0 \ 0.8 \ 1) \)

Exercise
Write a function \( \text{sum-square-difference n} \) that consumes a \( \text{Nat} \) 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.
\( \text{sum-square-difference} \ 3 \) \( \Rightarrow \ (- \ (\text{sqr} \ (+ \ 0 \ 1 \ 2 \ 3)) \ (+ \ 0 \ 1 \ 4 \ 9)) \) \( \Rightarrow \) 22

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

Exercise
Trace the program:

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

Exercise
Write a Racket function corresponding to
\[ g(x, y) = x \sqrt{x} + y^2 \]
\(\text{(sqrt n} \) computes \( \sqrt{n} \) and \( \text{(sqr n) computes n^2.})\)

Exercise
Complete tree-search. Clever bit: only search left or right, not both.

```scheme
;; (tree-search tree item) return #true if item is in tree.
;; tree-search: SSTree Num -> Bool
;; Example:
(check-expect (tree-search tree12 10) #true)
(check-expect (tree-search tree12 7) #false)
```
Exercise
Complete eval-binexp so it can handle `+` and `*`.

;; (eval-binexp expr) return the value of expr.
;; eval-binexp: BinExp -> Num
;; Examples:
(check-expect (eval-binexp (make-binode '* 7 6)) 42)
(check-expect (eval-binexp (make-binode '* 7 (make-binode '+ 4 2)))
42)

--

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)

Exercise
Complete the function (admission after5? age) that returns the admission cost.
;; admission: Bool Nat -> Num

Exercise
Create a function (even-mean-minus-odd-mean L) that returns the mean of the
even values in L minus the mean of the odd values.
Include a local helper function (mean M) that consumes a (listof Int) and re-
turns the mean of the values in M. Do not create any additional helper func-
tions.
(even-mean-minus-odd-mean (list 16 14 5 1)) => 12

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

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 (normalize L) that consumes a (listof Num), and returns the
list containing each value in L divided by the sum of the values in L. Compute
the sum only once.
(normalize (list 4 2 14)) => (list 0.2 0.1 0.7)

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
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)
Exercise

Given these definitions:

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

What is the value of this expression?

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

Exercise

Write a function that consumes a `Num`, and returns

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

Exercise

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

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

Exercise

Complete the function `polarcoord->posn` that consumes a `Polarcoord` and returns the `Posn` corresponding to the same point.

(Mathematically, $x = r \cos \theta$ and $y = r \sin \theta$.)

```scheme
;;; (polarcoord->posn p) convert p to rectangular coordinates
;;; polarcoord->posn: Polarcoord -> Posn
;;; Example:
;;; (check-within (polarcoord->posn (make-polarcoord 2 (/ pi 4)))
;;; (make-posn (sqrt 2) (sqrt 2)) 0.0001)
```

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

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)`.

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

Exercise

Complete `countdown-to` using recursion.

```scheme
;;; (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: Use `foldr` to write a function that behaves like `map`.

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

Exercise: Write a recursive function `vector-add` that adds two vectors.

\[(\text{vector-add (list 3 5) (list 7 11)} \Rightarrow \text{list 10 16})\]
\[(\text{vector-add (list 3 5 1 3) (list 2 2 9 3)} \Rightarrow \text{list 5 7 10 6})\]

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.

\[(\text{collatz-next 3}) \Rightarrow 10\]
\[(\text{collatz-next 12}) \Rightarrow 6\]

Exercise: Complete `sorted?`.

;;;; (sorted? L) return #true if every value in L is >= the one before.
;;;; sorted? (listof Int) -> Bool
;;;; Examples:

\[(\text{check-expect (sorted? (list)) #true})\]
\[(\text{check-expect (sorted? (list 2 3 3 5 7)) #true})\]
\[(\text{check-expect (sorted? (list 2 3 5 3 7)) #false})\]

What is the base case?

Exercise: Trace the program:

\[(+ \text{(remainder -10 2) (quotient 10 3)}) (* 2 3))\]

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

\[(\text{add-1-each (list 2 4 8)} \Rightarrow \text{list 3 5 9})\]

Exercise: Using recursion, create a function (and necessary helper functions) to create the times tables up to a given value. For example,

\[(\text{times-tables 4)} \Rightarrow \text{list (list 0 0 0 0)}\]
\[(\text{list 0 1 2 3})\]
\[(\text{list 0 2 4 6})\]
\[(\text{list 0 3 6 9})\]

Exercise: What is wrong with each of the following?

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

Exercise: Write a function that consumes a `(listof Nat)` and returns the list with each number doubled.

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

\[(\text{define (double-each L) (map (lambda (x) (+ x x)) L))}\]
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
Write a recursive function (step-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
(define x 4)
(define (f x) (* x x))
(f 3)

Exercise
Complete dot-product.

;; A Vector is a (listof Num).

;; (dot-produce u v) return the dot product of u and v.
;; dot-produce: Vector Vector -> Num
;; Requires: u and v have the same length.
;; Example:
;; (check-expect (dot-produce (list 2 3 5) (list 7 11 13)) 112)

Exercise
Complete dict-find. You may assume key appears at most once in dict.

;; (dict-find d key) return value associated with key in d.
;; If key is not in d, return #false.
;; dict-find: Dict Nat -> Any
;; Examples:
;; (check-expect (dict-find student-dict 7334) (make-student "Bill Gates" "appliedmath"))
;; (check-expect (dict-find student-dict 9999) #false)

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

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

Exercise
Complete merge.

;; (merge L1 L2) return the list of all items in L1 and L2, in order.
;; merge: (listof Num) (listof Num) -> (listof Num)
;; Requires: L1 is sorted; L2 is sorted.
;; Example:
;; (check-expect (merge (list 2 3 7) (list 4 6 8 9)) (list 2 3 4 6 7 8 9))
Exercise
Perform a trace of
$$(\text{or} \; (< 7 \; 4) \; (\leq 3 \; 3) \; (> 7 \; 4) \; (> 0 \; (/\; 3 \; 0)))$$

Exercise
Complete `flatten`. Hint: use the `append` function.

```lisp
;; (flatten tree) return the list of leaves in tree.
;; flatten: LLT -> (listof Num)
;; Examples:
(check-expect (flatten (list 1 (list 2 3) 4)) (list 1 2 3 4))
(check-expect (flatten (list 1 (list 2 (list 3 4)))) (list 1 2 3 4))
```

Exercise
Write a recursive function `(sum-between n b)` that consumes two `Nat`, with $n \geq b$, and returns the sum of all `Nat` between $b$ and $n$.

```lisp
(sum-between 5 3) \Rightarrow (+ 5 4 3) \Rightarrow 12
```

Exercise
Complete `factorize`. It may be helpful to consider the `count-up` template for recursion on a `Nat`.

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 recursive function `list-max` that consumes a nonempty `(listof Int)` and returns the largest value in the list.

Exercise
Following the template, complete `depth`.

```lisp
;; (depth tree) return the max distance from the root to a leaf of tree.
;; depth: LLT -> Nat
;; Examples:
(check-expect (depth (list 6 7)) 1)
(check-expect (depth (list 2 (list 3 (list 5)))) 3)
```

Exercise
Complete `count-sheep`.

```lisp
;; (count-sheep L) return the number of 'sheep in L.
;; count-sheep: (listof Any) -> Nat
;; Example:
(check-expect (count-sheep (list 6 'sheep 'ram 3.14 'sheep 'ox)) 2)
```

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.

```lisp
(countdown-by 15 3) \Rightarrow (list 15 12 9 6 3)
(countdown-by 14 3) \Rightarrow (list 14 11 8 5 2)
```
**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**

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

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

**Exercise**

Complete `insert`.

```
;; (insert item L) Add item to L so L remains sorted in increasing order.
;; ; ; insert: Int (listof Int) -> (listof Int)
;; ; ; Requires: L is sorted in increasing order.
;; ; ; Examples:
;; (check-expect (insert 6 (list 7 42)) (list 6 7 42))
;; (check-expect (insert 81 (list 3 9 27)) (list 3 9 27 81))
;; (check-expect (insert 5 (list 2 3 7)) (list 2 3 5 7))
```

**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 a function `(sum-odds-or-evens L)` that consumes a `(listof Int)`. If there are more evens than odds, the function returns the sum of the evens. Otherwise, it returns the sum of the odds.

Use `local`, but do not use L more than twice (in `map`, `filter`, `foldr`, or otherwise).

```
(sum-odds-or-evens (list 1 3 5 20 30)) => 9
```