Quiz!
write two versions of the same function

(define (*3–recursive L))
...

(define (*3–higher L))
...

(define 1)
(define 2)
(define 3)

(define 3)
(define 6)
(define 9)
Quiz!
write two versions of the same function

```
(define (*3-recursive L)
  (if (null? L)
      ()
      (cons (* 3 (first L)) (*3-recursive (rest L))))))
```
Quiz!
write two versions of the same function

(define (*3-recursive L)
  (if (null? L)
      '()
      (cons (* 3 (first L)) (*3-recursive (rest L))))

(define (*3-higher L)
  (map (lambda (n) (* 3 n)) L))
Racket miscellanea: symbols

```racket
Welcome to DrRacket, version 5.3.6 [3m].
Language: racket; memory limit: 128 MB.
> ella ; an undefined variable
 ella: undefined;
cannot reference an identifier before its definition
> 'ella ; a symbol!
 'ella
> '(ella 1917) ; a symbol in a list
 '(ella 1917)
> (symbol? (first '(ella 1917))) ; just to prove I wasn't lying
 #t
> (symbol? (second '(ella 1917))) ; an int is not a symbol
 #f
> (define ella 1917) ; defining a variable
 ella
> 'ella ; using a variable
 ella
> 'ella ; still a symbol!
 ella
> '(ella 1917) ; STILL a symbol in a list!
 '(ella 1917)
> (list ella 1917) ; it's a subtle difference
 '(1917 ella 1917)
```

docs.racket-lang.org/reference/symbols.html
Racket miscellanea: range & sort

Welcome to DrRacket, version 5.3.6 [3m].
Language: racket; memory limit: 128 MB.
> (define values (range 10))
> values
'(0 1 2 3 4 5 6 7 8 9)
> (sort values <)
'(0 1 2 3 4 5 6 7 8 9)
> (sort values >)
'(9 8 7 6 5 4 3 2 1 0)

docs.racket-lang.org/reference/symbols.html
Racket miscellanea: sort by key

docs.racket-lang.org/reference/pairs.html#(def._((lib._racket/private/list..rkt)._sort))
Racket miscellanea: member

Welcome to DrRacket, version 5.3.6 [3m].
Language: racket; memory limit: 128 MB.

```racket
> (range 10)
'(0 1 2 3 4 5 6 7 8 9)
> (member 10 (range 10))
#f
> (member 0 (range 10))
'(0 1 2 3 4 5 6 7 8 9)
> (member 5 (range 10))
'(5 6 7 8 9)
```

docs.racket-lang.org/reference/pairs.html#(def_.((lib_.racket/private/list..rkt)._member))
What does `matches` do?

(check-expect (matches '(ace 2 3 4) '(3 42 2)) 2)
(check-expect (matches '(ace 2 3 4) '(8 4 5)) 1)
(check-expect (matches '(ace 2 3 4) '(2 3 4 5)) 3)
How does `matches` do it?

```scheme
(define (matches L W)
  (length (filter (lambda (x) (member x W)) L)))

(check-expect (matches '(ace 2 3 4) '(3 42 2)) 2)
(check-expect (matches '(ace 2 3 4) '(8 4 5)) 1)
(check-expect (matches '(ace 2 3 4) '(2 3 4 5)) 3)
```

what happens if you change `filter` to `map`?
What Is a List?
Racket Lists
Inductive data structure, manipulated via constructors, selectors, and operations

**constructors**
*put together*

'()  
(cons <value> <list>)

**selectors**
take apart

(first <list>)
(rest <list>)
(null? <value>)

...  

**operations**
usually recursive

(length <list>)

...
What Is a Tree?
Trees

a unique path from the root to each node

root
(no parent)

height
length of longest path from root to leaf

leaves
(no children)
Binary Trees

*structure constraint:* every node has at most two children
Binary Search Trees (BSTs)

**order constraint:** every parent is greater than all the nodes in its left subtree and less than all the nodes in its right subtree.
Our BSTs
Inductive data structure, manipulated via constructors, selectors, and operations

### Constructors
*put together*

- `(make-empty-BST)`
- `(make-BST <key> <left> <right>)`

### Selectors
*take apart*

- `(emptyTree? <tree>)`
- `(key <tree>)`
- `(leftTree <tree>)`
- `(rightTree <tree>)`

### Operations
*usually recursive*

- `(size <tree>)`

...
Implementing BSTs

How would you encode a BST?

```
(define (make-empty-BST)
  '())

(define (make-BST key left right)
  (list key left right))
```

```
(define (make-BST key left right)
  (list key left right))
```
Implementing BSTs

How would you encode a BST?

(define (make-empty-BST)
  '())

(define (make-BST key left right)
  (list key left right))

```
  5
 / \
3   7
 / \
2   11
```
Implementing BSTs

How would you encode a BST?

(define (make-empty-BST)
  '())

(define (make-BST key left right)
  (list key left right))
Implementing BSTs

How would you encode a BST?

```
(define (make-empty-BST)
  '())

(define (make-BST key left right)
  (list key left right))
```

```
'(5
 (3
  (2 () ()))
  ()

(7
 ()
 (11 () ())))
```
Interface *vs.* Implementation
Interface *vs.* Implementation

\[
\text{size} = (\text{define} (\text{size} \ \text{tree}) \ \\
\quad (\text{if} (\text{emptyTree}? \ \text{tree}) \ \\
\quad \quad 0 \ \\
\quad \quad (+ 1 \ \\
\quad \quad (\text{size} \ (\text{leftTree} \ \text{tree}))) \ \\
\quad \quad (\text{size} \ (\text{rightTree} \ \text{tree})))))
\]
Interface vs. Implementation

(size)

(define (size tree)
  (if (emptyTree? tree)
      0
      (+ 1
         (size (leftTree tree))
         (size (rightTree tree)))))

(define (size tree)
  (if (null? tree)
      0
      (+ 1
         (size (second tree))
         (size (third tree)))))
What are some good cases?

(a) \( \varepsilon \) 

(b) \( k \) 

(c) \( k \rightarrow T_L \) 

(d) \( k \rightarrow T_R \) 

(e) \( k \rightarrow T_L, T_R \)
Deleting from a BST

\[ ( \text{delete-key } k_d \ T) \]

<table>
<thead>
<tr>
<th>( T )</th>
<th>( k_d = k )</th>
<th>( k_d \neq k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>( \varepsilon )</td>
<td>( \varepsilon )</td>
</tr>
<tr>
<td>( k )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_L )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( k )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_R )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( k )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

```
    5
   / \   \\
  3   7
 /     \
4   6
```
Deleting from a BST

(delete-key \( k_d \ T \))

<table>
<thead>
<tr>
<th></th>
<th>( k_d = k )</th>
<th>( k_d \neq k )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \varepsilon )</td>
<td>( \varepsilon )</td>
<td>( \varepsilon )</td>
</tr>
<tr>
<td>( k )</td>
<td>( \varepsilon )</td>
<td>( k )</td>
</tr>
<tr>
<td>( T_L )</td>
<td>( T_L )</td>
<td>(delete ( k_d \ T_L ))</td>
</tr>
<tr>
<td>( T_R )</td>
<td>( T_R )</td>
<td>(delete ( k_d \ T_R ))</td>
</tr>
</tbody>
</table>

2 cases...