tail position, tail recursion
Higher-order functions (1)
(+ 32 (* 1 (/ 9 5)))
(+ 32 (* 100 (/ 9 5)))
(+ 32 (* -2 (/ 9 5)))

(lambda (x)
  (+ 32 (* x (/ 9 5))))
(+ 32 (* 1 (/ 9 5)))
(- 32 (* 1 (/ 9 5)))
(* 32 (* 1 (/ 9 5)))

(lambda (x)
  (x 32 (* 1 (/ 9 5))))
Three famous list HOFs
Take a list of floats and round each one to two decimal places.

Take a list of strings and strip trailing whitespace.

Take a list of temperatures in Celsius and convert them to Farenheit.

Take a list of HTML elements and extract their attributes.
new_list = []
for x in lst:
    new_item = f(x)
    new_list.append(new_item)

print(new_list)
Take a list of floats and remove the ones < 50.

Take a list of strings and remove the ones that start with ‘a’.

Take a list of students and remove the ones in CSC324.

Take a list of HTML elements and remove all but the <a> tags.
new_list = []
for x in lst:
    if f(x):
        new_list.append(x)

print(new_list)
(define (func lst)
  (if (null? lst)
      0
      (+ (first lst)
          (func (rest lst))))
)

(define (func lst)
  (if (null? lst)
      null
      (append (func (rest lst))
              (first lst))))
Generic list recursion

(define (func f seed lst)
  (if (null? lst)
      seed
      (f (first lst)
        (func f seed (rest lst)))))
Generic list iteration

```
acc = seed
for x in lst:
    acc = f(x, acc)

print(acc)
```
Generic list iteration (done recursively)

(define (func f acc lst)
  (if (null? lst)
      acc
      (func f
               (func f
                        (f (first lst) acc)
                        (rest lst)))))
Summary: lookup the following functions in the Racket and Haskell docs

map
filter
foldl
foldr
Substitution demo
Laziness and foldl
Laziness and accumulators
Pattern-matching (I): value-based matching
\[ f(x) = \]
\[
\begin{align*}
\text{if } x &= 0 \\
&\quad \text{then} \\
&\quad \quad 10 \\
\text{else if } x &= 1 \\
&\quad \text{then} \\
&\quad \quad 20 \\
\text{else} \\
&\quad x + 30
\end{align*}
\]
Pattern-matching: structural
\[
g \ lst = \\
\text{if null } \ lst \\
\text{then} \\
\text{10} \\
\text{else} \\
\text{let } x = \text{head } \ lst \\
\text{xs} = \text{tail } \ lst \\
\text{in} \\
x + \text{length } \ xs
\]

\[
g \ [ ] = 10 \\
g \ (x:xs) = \\
x + \text{length } \ xs
\]
Higher-order functions (2)
(+ 32 (* 1 (/ 9 5)))
(+ 32 (* 100 (/ 9 5)))
(+ 32 (* -2 (/ 9 5)))

(lambda (x)
  (+ 32 (* x (/ 9 5))))
(lambda (n) (+ n 1))

(lamba (n) (+ n 200))

(lamba (n) (+ n -324))

(lamba (x)
    (lambda (n) (+ n x)))
Implementation of returned functions
(define (make-f n)
    (lambda (x)
      ; REALLY long body
      (... n ... x ...)))

(define f-1 (make-f 1))
(define f-2 (make-f 2))
(define f-3 (make-f 3))
environment:
a mapping of identifiers to values
Key idea:

1. store the body once
2. save the environment separately for each of
   \((\text{make-f } 1), (\text{make-f } 2), (\text{make-f } 3)\)