A useful list function is one that concatenates two lists into a single list. Often, this function is called `append`. Consider these recurrence relations for `append`:

append list a to list b: place the head of list a onto the append of the tail of list a and the list b

append the empty list to list b: list b

These relations can be directly translated into Scheme:

```
(define (append a b)
  (if (not a)
      b
      (cons (car a) (append (cdr a) b)))
)
```

Once `append` is in place, we can use it to write `reverse`:

```
(define (reverse a)
  (if a
      (append (reverse (cdr a)) (list (car a)))
      nil)
)
```

Note that the variable `nil` is often used to represent the empty cons object/list. Surprisingly, the iterative form of reverse is simpler (in that it only uses `cons`).

```
(define (reverse a)
  (define (_reverse supply result)
    (if (supply)
        (_reverse (cdr supply) (cons (car supply) result))
        result)
  )

(_reverse a ())
```

Why can we get away with using `cons` in this case?