Exercise 2.27.

Modify your reverse procedure of exercise 2.18 to produce a deep-reverse procedure that takes a list as argument and returns as its value the list with its elements reversed and with all sublists deep-reversed as well. For example,

```
(define x (list (list 1 2) (list 3 4)))

x
((1 2) (3 4))

(reverse x)
((3 4) (1 2))

(deep-reverse x)
((4 3) (2 1))
```

Answer.

To implement deep-reverse, recall the recursive plan for computing reverse:

- Reverse of the empty list is nil.

- Else, if the list contains only one element, then just return the list wholly intact.

- Otherwise, reverse all but the last element of the list, and cons that last element onto the result.

Deep-reverse is similar, the value of the empty list is the same:

- Deep-reverse of empty list is nil.

But in the reduction step, where we extract the former sublists and the last sublist of the list, we must take into account the case where the sublists may themselves made up of lists that we need to reverse. Thus, the appropriate reduction step is

- Deep-reverse of a list where we cons the deep-reverse of its last sublist onto the deep-reverse of the former sublists of it.

Finally, by successively taking sublist we reach the elements, so we need another base case:

- Deep-reverse of a number is the number itself.

Thus, this reveals the complete procedure:

```
(define (deep-reverse items)
  (cond ((null? items) nil)
        ((not (pair? items)) items)
        (else
          (cons (deep-reverse (last-sublist items))
                (deep-reverse (former-sublists items))))))
```

where the procedures last-sublist and former-sublists are both almost identical to the last-element and former-elements in exercise 2.18:[1]

[1]. Similar to exercise 2.18, we can also implement the procedure last-sublist as:

```
(define (last-sublist items)
  (cond ((null? items) nil)
        ((null? (cdr items)) (car items))
        (else
          (last-sublist (cdr items)))))
```

```
(define (last-sublist items)
  (let ((list-length (length items)))
    (list-ref items (- list-length 1))))

(define (former-sublist items)
  (if (or (null? items)
          (null? (cdr items)))
      nil
      (cons (car items)
            (former-sublists (cdr items)))))
```