

Exercise 2.18.

Define a procedure `reverse` that takes a list as argument and returns a list of the same elements in reverse order:

```
(reverse (list 1 4 9 16 25))
(25 16 9 4 1)
```

Answer.

To `reverse` a list, do the following:

- 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.

Using this strategy, we can write down the procedure `reverse`:

```
(define (reverse items)
  (cond ((null? items) nil)
        ((null? (cdr items)) items)
        (else
         (cons (last-element items)
               (reverse (former-elements items))))))
```

One of the auxiliary procedure `last-element` which picks the last element in a given list can either be implemented as


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

or to be

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

And the other one, the `former-elements`, constructs a list that contains all but the last element of the original list

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

*. Creative Commons  2013, Lawrence X. Amlord (颜世敏, aka 颜序).