Exercise 2.19.

Consider the change-counting program of section 1.2.2. It would be nice to be able to easily change the currency used by the program, so that we could compute the number of ways to change a British pound, for example. As the program is written, the knowledge of the currency is distributed partly into the procedure `first-denomination` and partly into the procedure `count-change` (which knows that there are five kinds of U.S. coins). It would be nicer to be able to supply a list of coins to be used for making change.

We want to rewrite the procedure `cc` so that its second argument is a list of the values of the coins to use rather than an integer specifying which coins to use. We could then have lists that defined each kind of currency:

```
(define us-coins (list 50 25 10 5 1))
(define uk-coins (list 100 50 20 10 5 2 1 0.5))
```

We could then call `cc` as follows:

```
(cc 100 us-coins)
292
```

To do this will require changing the program `cc` somewhat. It will still have the same form, but it will access its second argument differently, as follows:

```
(define (cc amount coin-values)
  (cond ((= amount 0) 1)
        ((or (< amount 0) (no-more? coin-values)) 0)
        (else
          (+ (cc amount
                 (except-first-denomination coin-values))
             (cc (- amount
                    (first-denomination coin-values))
                 coin-values)))))
```

Define the procedures `first-denomination`, `except-first-denomination`, and `no-more?` in terms of primitive operations on list structures. Does the order of the list `coin-values` affect the answer produced by `cc`? Why or why not?

Answer.

The procedures `first-denomination`, `except-first-denomination`, and `no-more?` are implemented in terms of primitive operations on list structures as follow:

```
(define (first-denomination coin-values)
  (car coin-values))

(define (except-first-denomination coin-values)
  (cdr coin-values))

(define (no-more? coin-values)
  (null? coin-values))
```

The order of the list `coin-values` does not affect the answer produced by `cc`. This can be verified by running `cc` on a couple of lists and their reverse:

```
(cc 100 us-coins)
;Value: 292

(cc 100 (reverse us-coins))
;Value: 292
```

```
(cc 100 uk-coins)
;Value: 104561

(cc 100 (reverse uk-coins))
;Value: 104561
```