The binary function `map` takes a unary function and a list.

```
(map sqr (list 2 0 1))  ; Make a list of the squares of 2 0 1 and 5.
```

Step: the effect of `map`.

```
(list (sqr 2) (sqr 0) (sqr 1) (sqr 5))
```

Unfortunately, the stepper doesn't show that step.

The remaining steps you're welcome to do all at once when asked.

```
(list 4 (sqr 0) (sqr 1) (sqr 5))
(list 4 0 (sqr 1) (sqr 5))
(list 4 0 1 (sqr 5))
(list 4 0 1 25)
```

Summary of the result:

```
(check-expect (map sqr (list 2 0 1 5))
  (list 4 0 1 25))
```

Intermediate Step

```
The intermediate step for `map` [which isn't shown in the stepper] is:
  (map f (list a b c ...))
  (list (f a) (f b) (f c) ...)
```

EXAMPLE.

From a named list of the natural numbers from 1 to 7 inclusive,
produce a list of the squares of those numbers.

```
(define 1-to-7 (range 1 8 1))
(map sqr 1-to-7)  ; Square each element in 1-to-7 individually.
(map sqr (list 1 2 3 4 5 6 7))
(list (sqr 1) (sqr 2) (sqr 3) (sqr 4) (sqr 5) (sqr 6) (sqr 7))
(list 1 4 9 16 25 36 49)
```

Summary of the result:

```
(check-expect (map sqr 1-to-7)
  (list 1 4 9 16 25 36 49))
```
; EXAMPLE.
; Let's produce a list of images from a list of numbers.
(require picturing-programs)
;
; Use 'map' with:
;  • a unary function that produces an image from a number
;  • a list of numbers
;
; Let's make our own function that produces an image from a number.
;
; tall-ellipse : number -> image
; A solid maroon coloured ellipse 10 pixels wide and 10 x n pixels high.
(define (tall-ellipse n)
  (ellipse 10 (* 10 n) "solid" "maroon"))
;
(check-expect (tall-ellipse 3))
;
; Now map it onto that earlier list of numbers.
(map tall-ellipse 1-to-7)
(map tall-ellipse (list 1 2 3 4 5 6 7))
(list (tall-ellipse 1)
  (tall-ellipse 2)
  (tall-ellipse 3)
  (tall-ellipse 4)
  (tall-ellipse 5)
  (tall-ellipse 6)
  (tall-ellipse 7))

(list    )
;
; Summary of the result:
(check-expect (map tall-ellipse 1-to-7)
(list    ))