

Exercise 1.46.

Several of the numerical methods described in this chapter are instances of an extremely general computational strategy known as *iterative improvement*. Iterative improvement says that, to compute something, we start with an initial guess for the answer, test if the guess is good enough, and otherwise improve the guess and continue the process using the improved guess as the new guess. Write a procedure `iterative-improve` that takes two procedures as arguments: a method for telling whether a guess is good enough and a method for improving a guess. `Iterative-improve` should return as its value a procedure that takes a guess as argument and keeps improving the guess until it is good enough. Rewrite the `sqrt` procedure of section 1.1.7 and the `fixed-point` procedure of section 1.3.3 in terms of `iterative-improve`.

Answer.

The procedure `iterative-improve` here is very similar to `fixed-point` in section 1.3.3.


```
(define (iterative-improve close-enough? improve)
  (lambda (x)
    (define (try guess)
      (let ((next (improve guess)))
        (if (close-enough? guess next)
            next
            (iter next))))
    (try x)))

(define (close-enough? v1 v2)
  (let ((tolerance 0.00001))
    (< (abs (- v1 v2)) tolerance)))
```

With the aid of `iterative-improve`, we can express `sqrt` and `fixed-point` procedure much more concisely:

```
(define (sqrt x)
  ((iterative-improve close-enough?
                      (average-damp (lambda (y) (/ x y))))
   1.0))

(define (fixed-point f first-guess)
  ((iterative-improve close-enough? f) first-guess))
```

*. Creative Commons  2013, Lawrence R. Amlord(颜世敏).