Recursive vs Iterative Processes
Decide whether each of the following procedures generates a recursive or iterative process.

(define (foo x)  
  (* (- (+ (/ x 3) 4) 6) 2))
Neither

(define (foo x)  
  (if (= x 0) 0 (+ x (foo (- x 1))))
Recursive

(define (helper1 x)  
  (if (= x 0) 1 (helper1 (- x 1))))
Iterative

(define (helper2 x)  
  (if (= x 0) 1 (+ 1 (helper2 (- x 1)))))
Recursive

(define (bar x)  
  (if (even? x) (helper1 (- x 1)) (helper1 (- x 2))))
Iterative

(define (bar x)  
  (if (even? x) (helper2 (- x 1)) (helper2 (- x 2))))
Recursive

(define (bar x)  
  (if (= x 0) (helper2 x) (helper1 x)))
Iterative

(define (bar x)  
  (if (= x 0) (helper2 x) (helper2 x)))
Recursive

(define (bar x)  
  (cond ((= x 0) 1)  
        ((= (helper2 x) 3) 5)  
        (else (helper1 x)))))
Recursive

(define (bar x)  
  (helper2 (helper1 x)))
Recursive

More Recursion Practice
1. There used to be a number 1. Now there isn’t, but I didn’t want to renumber all the questions, so here’s a placeholder.

2. There is something called a falling factorial. (falling n k) means that k consecutive numbers should be multiplied together, starting from n and working downward. For example, (falling 7 3) means 7 * 6 * 5. Write the procedure falling that generates an iterative process.

(define (falling b n)  
  (define (helper b n ans)  
    (if (= n 1)  
        (* b ans)  
        (helper (- b 1) (- n 1) (* b ans))))
3. Write a version of (expt base power) that works with negative powers as well.

(define (expt base power)
  (cond ((= power 0) 1)
        ((> power 0) (* base (expt base (- power 1))))
        (else (/ (expt base (+ power 1)) base))))

4. Implement (ab+c a b c) that takes in values a, b, c and returns (a*b) + c. However, you cannot use *. Make it a recursive process.

(define (ab+c a b c)
  (if (= b 0)
      c
      (+ a (ab+c a (- b 1) c))))

5. Implement (ab+c a b c) as an iterative process. Don't define helper procedures.

(define (ab+c a b c)
  (if (= b 0)
      c
      (ab+c a (- b 1) (+ c a))))

Order of Growth

Decide what the order of growth is for each of the following

(define (fact x)
  (if (= x 0)
      1
      (* x (fact (- x 1))))
  O(n)

(define (fact-iter x answer)
  (if (= x 0)
      answer
      (fact-iter (- x 1) (* answer x))))
  O(n)

(define (sum-of-facts x n)
  (if (= n 0)
      0
      (+ (fact x) (sum-of-facts x (- n 1))))
  O(nx)

(define (fib n)
  (if (<= n 1)
      1
      (+ (fib (- n 1)) (fib (- n 2))))
  O(2^n)

(define (square n)
  (cond ((= n 0) 0)
        (even? n) (* (square (quotient n 2)) 4))
        (else (+ (square (- n 1)) (- (+ n n) 1)))))
  O(log n)