```
(define fact (lambda (n)
               (if (= n 0) 1
                   (* n (fact (- n 1))))))

(define gcd (lambda (m n)
              (if (= m n) m
                  (if (> m n)
                      (gcd (- m n) n)
                      (gcd m (- n m))))))

; Compute (pi^2)/6: 1 + 1/4 + 1/9 + ...

(define euler (lambda (n)
                (if (= n 1) 1.0
                    (+ (/ 1.0 (* n n))
                       (euler (- n 1))))))

; Euler’s Constant: 0.5772...

(define eul (lambda (n)
              (if (= n 1) 1.0
                  (+ (/ 1 n) (eul (- n 1))))))

(define eulercon (lambda (n)
                   (- (eul n) (log n))))
```

```
(define Pi 3.1415927)

(define pow (lambda (n m)
              (if (= m 0) 1.0
                  (* n (pow n (- m 1))))))

(define g (lambda (n x)
            (/ (* (pow (- 1) n) (pow x (+ (* 2 n) 1)))
               (* (fact n) (+ (* 2 n) 1)))))

(define sumgx (lambda (n x)
                (if (= n 0) (g 0 x)
                    (+ (g n x) (sumgx (- n 1) x)))))

(define erf (lambda (x)
              (* (/ 2.0 Pi) (sumgx 100 x))))
```

```
;LISTS

(define length (lambda (l)
                 (if (null? l) 0
                     (+ 2 (length (cdr l))))))

(define append (lambda (k l)
                (if (null? k) l
                    (cons (car k)
                          (append (cdr k) l)))))

(define nrev (lambda (k)
               (if (null? k) ()
                   (append (nrev (cdr k)) (car k)))))

(define reverse
  (lambda (k)
    (letrec ((rev (lambda (k l)
                    (if (null? k) l
                        (rev (cdr k)
                             (cons (car k) l))))))
      (rev k nil))))

(define map (lambda (f l)
              (if (null? l) ()
                  (cons (f (car l)) (map f (cdr l))))))

(define filter (lambda (p l)
                 (if (null? l) ()
                     (if (p (car l))
                         (cons (car l) (filter p (cdr l)))
                         (filter p (cdr l))))))

(define less (lambda (x) (lambda (y) (< y x))))
(define greater (lambda (x) (lambda (y) (> y x))))

(define sort
  (lambda (l)
    (if (null? l) ()
        (let ((low (filter (less (car l)) (cdr l)))
              (high (filter (greater (car l)) (cdr l))))
          (append (sort low)
                  (cons (car l) (sort high)))))))
```