```
(defun gcd (n m)
  "Function to determine the greatest common divisor"
  (if (> n m)
      (let ((dividend n) (divisor m) (remainder 1))
        (while (not (= remainder 0))
          (setq remainder (% dividend divisor))
          (setq dividend divisor)
          (setq divisor remainder))
        dividend)
    (gcd m n)))


(setq L '(3 1 5 6 3 2 3))
(3 1 5 6 3 2 3)

(setq z 3)
3

(setq y 9)
9

(defun replace (L z y)
  (setq k (lambda (x) (if (= x z) y x)))
  (mapcar k L))
replace

(replace L z y)
(9 1 5 6 9 2 9)

(defun print-list (L)
  (mapc (lambda (x) (princ x) (princ " ")) L))
print-list
-
(replace L z y) ; (9 1 5 6 9 2 9)

(defun print-list (L)
  (mapc (lambda (x) (princ x) (princ " ")) L))
print-list

; Testing function: print-list
(print-list L) ; 3 1 5 6 3 2 3 (3 1 5 6 3 2 3)

; Testing function: print-list (w/ funcall)
(funcall 'print-list L) ; 3 1 5 6 3 2 3 (3 1 5 6 3 2 3)

; Testing function: replace (w/ apply)
(apply 'replace (list L z y)) ; (9 1 5 6 9 2 9)

; Ex. 3 - List Manipulation
; Ex. 3a - Function: make-multiples
(defun make-multiples (n m)
  (if (and (> n 0) (> m 0))
```

```
      (let ((L '()) (mult m) (limit m))
        (dotimes (count limit result)
          (setq result (* n mult))
          (push result L)
          (setq mult (1- mult)))
        L)
    nil))
make-multiples

; Testing function: make-multiples
(make-multiples 4 5) ; (4 8 12 16 20)

; Ex. 3b - Function: is-multiple
(setq L '(4 8 12 16))
(4 8 12 16)

(3 5 7 9)

(3 6 9 12)


(defun is-multiple (L)
  (cond
   ((not L) nil)
   ((not (cdr L)) t)
   (t (let ((mult 2) (i (car L)))
        (dotimes (count (1- (length L)) result)
          (setq j (car (cdr L)))
          (if (= j (* i mult)) (setq result t)
            (setq result nil)
            (pop L)
            (setq mult (1+ mult))))
        result))))
is-multiple

(is-multiple L)
t
```