```
#lang racket

;; Last edit: Mar. 28, 2017, dlb


(define (racket-1)
  (newline)
  (display "Racket-1: ")
  (flush-output)
  (print (eval-1 (read)))
  (newline)
  (racket-1)
  )




;Contains new code for assignment
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (eval-1 exp)
  ; display expressions added. -- dlb
  (display "Executing eval-1, exp = ")
  (displayln exp)

  (cond ((constant? exp)
         (displayln "constant? is true")
         exp)

        ((symbol? exp)
         (displayln "symbol? is true")
         (eval exp))  ; use underlying Racket's EVAL

        ((quote-exp? exp) (quote-helper exp)) ;checks if too many
args


        ((if-exp? exp)
         (displayln "if-exp? is true")
         (if (eval-1 (cadr exp))   ; use underlying Racket's IF
             (eval-1 (caddr exp))
             (eval-1 (cadddr exp))))

        ((lambda-exp? exp)
         (displayln "lambda-exp? is true")
         exp)
        ((map-exp? exp)  ;part of ex6
         (map-1 (eval(cadr exp)) (eval (caddr exp))))



        ((and-exp? exp)   ;part of ex7
```

```
          (cond
            ((=(length exp)1) '#t)
            ((=(length exp) 2) (eval-1 (cadr exp)))
            ((equal? (cadr exp) '#t)(eval-1 (caddr exp)))
            ((equal? (cadr exp) '#f) '#f)
            ((equal? (eval-1(cadr exp)) '#f)'#f)
            ((equal? (eval-1(cadr exp)) '#t)(eval-1(caddr exp)))
            ))


         ((pair? exp)
          (displayln "pair? is true")
          (apply-1 (eval-1 (car exp))      ; eval the operator
                   (map eval-1 (cdr exp))))

         (else (error "bad expr: " exp))))



 (define (map-1 proc arg) ;map-1 so that racket doesnt use map ex6
   (map proc arg))


 (define (quote-helper arg) ;we can only have one argument for quote
(ex5)
   (if (> (length arg) 2) (error "Too many arguments" arg) (cadr
arg)))




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Code that came with the file
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(define (apply-1 proc args)
  ; display expressions added. -- dlb
  (display "Executing apply-1, proc = ")
  (display proc)
  (display " args = ")
  (displayln args)

  (cond ((procedure? proc)  ; use underlying Racket's APPLY
         (displayln "procedure? is true")
         (apply proc args))

        ((lambda-exp? proc)
         (displayln "lambda-exp? is true")
```

```
          (let ([s (substitute (caddr proc)   ; the body
                               (cadr proc)    ; the formal parameters
                               args           ; the actual arguments
                               '())])         ; bound-vars, see below
            (begin
              (display "substitute returned ")
              (displayln s)
              (eval-1 s))))

        (else (error "bad proc: " proc))))




;; Some trivial helper procedures:

(define (constant? exp)
  (or (number? exp) (boolean? exp) (string? exp) (procedure? exp)))

(define (exp-checker type)
  (lambda (exp) (and (pair? exp) (eq? (car exp) type))))


;Contains new code for assignment
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define quote-exp? (exp-checker 'quote))
(define if-exp? (exp-checker 'if))
(define lambda-exp? (exp-checker 'lambda))
(define map-exp? (exp-checker 'map-1));helper for map-1
(define and-exp? (exp-checker 'and)); helper for and
;;;;;;;;;;;;;;;;;;;;


(define (substitute exp params args bound)
  ; display expressions added. -- dlb
  (display "Executing substitute, exp = ")
  (display exp)
  (display " params = ")
  (display params)
  (display " args = ")
  (display args)
  (display " bound = ")
  (displayln bound)

  (cond ((constant? exp) exp)
        ((symbol? exp)
         (if (memq exp bound)
             exp
             (lookup exp params args)))
        ((quote-exp? exp) exp)
```

```
        ((lambda-exp? exp)
         (list 'lambda
               (cadr exp)
               (substitute (caddr exp) params args (append bound
(cadr exp)))))
        (else (map (lambda (subexp) (substitute subexp params args
bound))
                   exp))))

(define (lookup name params args)
  (cond ((null? params) name)
        ((eq? name (car params)) (maybe-quote (car args)))
        (else (lookup name (cdr params) (cdr args)))))

(define (maybe-quote value)
  (cond ((lambda-exp? value) value)
        ((constant? value) value)
        ((procedure? value) value)  ; real Racket primitive procedure
        (else (list 'quote value))))
```