Question 1. Assume the following two definitions have been entered/run ...

\[
\text{(define L (list "A"}
\begin{array}{l}
\quad \text{(list (list "B" "C") "D" "E")}
\quad \text{(list}
\quad \quad \text{(list "F" (list "G" "H")))})
\end{array}
\]

Evaluate the following three expressions, and show the two sets of steps (with the usual underlining) ...

\[
\begin{align*}
\text{(length L)} & \quad \text{(reverse L)} \\
\text{(select text? L)} & \\
\text{(step (map reverse LL))} & \quad \text{(step (map length LL))}
\end{align*}
\]

Now fill in the blanks to make these tests pass. Use expressions that call only the functions first, second, third, and/or fourth, and use the variable L, but use no literal texts.

\[
\begin{array}{ll}
\text{(same! "A" } & \text{ (same! "B")} \\
\text{(same! "C" } & \text{ (same! "D")} \\
\text{(same! "E" } & \text{ (same! "F")} \\
\text{(same! "G" } & \text{ (same! "H")}
\end{array}
\]
Question 2. The function join takes any number of lists and produces a list containing all the elements of the lists, in the same order ...

(same! (join (list "A" "B") (list "C") (list "D" "E" "F")))

: Evaluate each of the following expressions. If an expression produces an error, just write “error”.

(join (list "A" "B") (list "C") (list "D" "E" "F"))

(join (list "A") (list (list "B" "C") "D") (list "E") (list "F") (list "G" "H")))

(join "A" "B")

(join (list "A" "B"))

(join (list (list "A" "B") (list "C") (list "D" "E" "F")))

(join)

: Question 3. Assume the following definition has been entered/run ...

(define (>: v
    (if (list? v) (map :> v)
        else :v )))

: ... and show the following two sets of steps (you may omit the underlining for the steps) and the values of the two expressions ...

(step (hide :>) (>: "A"))

(step (hide :>) (>: (list "B" "C")))

(>: (list (list "B" "C") "D"))

(>: (list "A" (list (list "B" "C") "D") "E"))
: Question 4. Based on the following two definitions, show the following sets of steps (you may omit the underlining for the steps) and the values of the expressions ... 

(define (N v)
  (if (list? v) (combine + (map N v))
      else 1))

(step (hide N) (N "A"))

(define (:: v)
  (if (list? v) (combine join (map :: v))
      else (list 😊)))

(step (hide N) (N (list "B" "C")))

N (list (list "B" "C") "D")

N (list "A" (list (list "B" "C") "D") "E")

(step (hide ::) (:: "A"))

N (list (list "B" "C") "D")

:: (list (list "B" "C") "D")

:: (list "A" (list (list "B" "C") "D") "E")