```
;; Function To Be Tested: read-from-string
;;
;; Source:        CLtL p. 380
;;
;; Chapter 22: Input/Output         Section 2.1: Input from Character Streams
;;
;; Created By:    Peter Reidy
;;
;; Creation Date: 16 November 86
;;
;; Last Update: 22 January 87
;;
;; Filed As:      {eris}<lispcore>cml>test>22-2-1-read-from-string.test
;;
;; Syntax: read-from-string string  &optional eof-effor-p eof-value
;;                                         &key :start :end :preserve-
whitespace
;;
;; Function Description: reads the part of string delimited by :start and :end
and returns the lisp object built by the reader from it.
;;
;; Arguments
;;    string: a string
;;    eof-error-p, eof-value: if true, return an error at eof; otherwise,
return the value of eof-value
;;    :start, :end: (counting from 0) delimiters of the portion of the string
to read
;;    :preserve-whitespace: if true, reads whitespace characters as
syntactically significant.
;;
;; Returns: the object created by the reader, and the length of the string

;;
(do-test-group read-from-string-group
      :before
      (progn
            (test-setq  symbol5 '|5|
                        symbol55 '\5\5
                        list5 (list 5 4 3 2 1)
                        \5 6
                        |55| 66
            )
      )  ; progn
      (do-test "read-from-string produces symbols"
            (and
                  (every 'equal
                        (list (read-from-string "|5|") (read-from-string
"|55|"))
                        (list symbol5 symbol55)
                  )
                  (every '=
                        (list (eval (read-from-string "|5|")) (eval (read-
from-string "|55|")))
                        '(6 66)
                  )
            )  ; and
      )  ; do-test "read-from-string produces symbols"
;;
      (do-test "read-from-string produces strings"
            (string-equal "Alexis is a bitch." (read-from-string "\"Alexis is
a BITCH.\""))
      )  ; do-test "read-from-string produces strings"
;;
```

```
      (do-test "read-from-string produces lists"
            (and
                  (listp (eval (read-from-string      "list5")))
                  (listp (read-from-string "(5 4 3 2 1)"))
                  (= 1 (car (last (eval (read-from-string "list5")))))
            )  ; and
      )  ; do-test "read-from-string produces lists"
;;
      (do-test "read-from-string length value"
            ;; the object read is the same, but the strings' lengths are
different.
            (let ((version1 "(+ 3 3)") (version2 "(    + 3
3  )"))
                  (and
                        (equal
                              (car (multiple-value-list (read-from-string
version1)))
                              (car (multiple-value-list (read-from-string
version2)))
                        )  ; equal
                        (not (equal (cadr (multiple-value-list (read-from-
string version1)))
                                          (cadr (multiple-value-list (read-
from-string version2)))
                        ))  ; not equal
                  )  ; and
            )  ; let
      )  ; do-test "read-from-string length value"
;;
      (do-test "read-from-string start and end keywords"
            (every #'(lambda (arg) (= (read-from-string "123") arg))
                  (list (read-from-string "0123" nil nil :start 1 :end 4)
                        (read-from-string "1234" nil nil :end 3)
                        (read-from-string "01234" nil nil :start 1 :end 4)
                        (read-from-string "01234" nil nil :end 4 :start 1)
                  )
            )  ; every
      )  ; do-test "read-from-string start and end keywords"
;;
      (do-test "read-from-string returns evaluable expressions"
            (and
                  (= 6 (eval (read-from-string "(+ 3 3)")))
                  (= 6 (eval (read-from-string "xxx(+ 3 3)!!!" nil nil :start
3 :end 10)))
                  (= (eval (read-from-string "(+ 3 3)")) (eval (read-from-
string "xxx(+ 3 3)!!!" nil nil :start 3 :end 10)))
            )  ; and
      )  ; do-test "read-from-string returns evaluable expressions"
;;
      (do-test "read-from-string eof arguments"
            (and
                  (= 0 (read-from-string "      " nil 0))
                  (expect-errors (error) (read-from-string "(car (list 1 2 3)"
t 0))
            )  ; and
      )  ; do-test "read-from-string eof arguments"
)  ; do-test-group
STOP
```