DESCRIPTION: evaluating, testing, and displaying different types of data expressions.

WRITE YOUR NAME IN PLACE OF "TYPE YOUR NAME HERE" IN THE DISPLAY STATEMENT GIVEN BELOW.

(display "
==========================================
CMPU-101, Fall 2018
Lab-1
SOLUTIONS
==========================================
"
(newline)
;; For problems 1-3, write the requested expressions in "check-expect" statements. Then include the expression as a string in a display statement (as shown below in an example).
;; Insert one or more \n's at the beginning or end of the string in each new display statement you write. Doing so adds some blank lines between non-blank ones. Another way to do this without embedding any escape sequence in the output string is to use the newline procedure on a line by itself like this: (newline)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(display "Problem 1: Functions using numbers\n") ; \n for newline
;;
;; --EXAMPLE: the following expression adds numbers 40, 1, and 1
;; UNCOMMENT EACH OF THE 4 LINES BELOW TO SEE THE RESULT
;;
(check-expect (+ 40 1 1) 42) ; test expression
(display "(+ 40 1 1) ==> ") ; display expression
(+ 40 1 1) ; evaluate expression
(newline) ; blank line in output

(display "\n(1a) Evaluate an expression to add numbers 1 to 5 \n")
;; Write a check-expect statement to show that you know what the function should produce, include the expression you wrote in a display statement in a string, and then write the expression for evaluation.
(check-expect (+ 1 2 3 4 5) 15) ; test expression
(display "(+ 1 2 3 4 5): ") ; display expression
(+ 1 2 3 4 5) ; evaluate expression
(newline) ; blank line in output
(display "\n(1b) Evaluate 42 - 11 - 5 - 10 \n")
;; Write the expression above in prefix notation, test it
;; with check-expect, include the expression you wrote in a
;; display statement, and then write the expression for
;; evaluation.
(check-expect (- 42 11 5 10) 16); test expression
(display "(- 42 11 5 10): ") ; display expression
(- 42 11 5 10) ; evaluate expression
(newline) ; blank line in output

(display "\n(1c) Evaluate (3 + 4) * (10 / 2) \n")
;; Write the expression above in prefix notation, test it
;; with check-expect, include the expression you wrote in a
;; display statement, and then write the expression for
;; evaluation.
(check-expect (* (+ 3 4) (/ 10 2)) 35); test expression
(display "(* (+ 3 4) (/ 10 2)): ") ; display expression
(* (+ 3 4) (/ 10 2)) ; evaluate expression
(newline) ; blank line in output

;;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++
(display "\n\nProblem 2: Predicate Functions\n")

(display "\n(2a) Evaluate (> 42 9 4 4)\n")
;; Write the expression above that compares 4 numbers
;; in a check-expect statement, include the expression
;; in a display statement, and then evaluate the
;; expression:
(check-expect (> 42 9 4 4) false); test expression
(display "(> 42 9 4 4): ") ; display expression
(> 42 9 4 4) ; evaluate expression
(newline) ; blank line in output

(display "\n(2b) Evaluate ((4 >= 4) and (5 < 10))\n")
;; Write the expression above in prefix notation
;; in a check-expect statement, include the expression
;; in a display statement, and then evaluate the
;; expression:
(check-expect (and (>= 4 4) (< 5 10)) true); test expression
(display "(and (>= 4 4) (< 5 10)): ") ; display expression
(and (>= 4 4) (< 5 10)) ; evaluate expression
(newline) ; blank line in output

(display "\n(2c) Evaluate ((2 = 3) or (7 > 9))\n")

Write the expression above in prefix notation
in a check-expect statement, include the expression
in a display statement, and then evaluate the
expression:
(check-expect (or (= 2 3) (> 7 9)) false) ; test expression
(display "(or (= 2 3) (> 7 9)): ") ; display expression
(or (= 2 3) (> 7 9)) ; evaluate expression
(newline) ; blank line in output

PRESS RUN TO SEE THE RESULTS OF ALL CHECK-EXPECTS AND
DISPLAY STATEMENTS FOR PROBLEMS 1 AND 2 IN THE IW.
A LINE AT THE BOTTOM OF THE IW SHOULD SAY: ALL 7 TESTS PASSED!

Problem 3: Writing predicate function between?

DESIGN RECIPE FOR FUNCTIONS:

1. Write a contract and purpose statement. The contract should include the actual parameter names:
2. Write sufficient test cases for this function. Make sure these cases use inputs that cover the input domain.
3. Write the function header, including parameter names.
   Write the function body to calculate the desired output using the input parameter names and whatever primitive functions you need.
4. Run the function, making sure that all tests pass and display all function calls.

Write a function called "between?" that takes three numeric inputs a, x, and y, and evaluates to #t (true) if a is between x and y (including the equal cases) and #f (false) otherwise.
Remember, you don't know if x > y or y > x.
Be sure to follow the design recipe. Write the function, write at least 4 check-expect statements, and then write a display statement containing the function definition to write the function to the IW. The contract, inputs and purpose are already written in the 3 lines below. Just un-comment the next 5 lines.
CONTRACT: (between? a x y) -> boolean
INPUT: a, x, and y are numbers
OUTPUT: a boolean, #t if a is between x and y and #f otherwise

Write 4 check-expect statements below this line:
(check-expect (between? 5 4 8) #t)
(check-expect (between? 5 5 5) #t)
(check-expect (between? 55 4 8) #f)
(check-expect (between? -5 5 5) #t)

Define the function below this line.
(define (between? a x y)
  (or
   (and (>= a x) (<= a y)) ;; x < y
   (and (>= a y) (<= a x)) ;; y < x
  )))

Display the function definition below this line.
(display "(define (between? a x y)
  (or
   (and (>= a x) (<= a y)) ;; x < y
   (and (>= a y) (<= a x)) ;; y < x
  ))")

Display the result of calling this function below this line.
(display "(between? 13 12 14): ")
(between? 13 12 14)

... PRESS RUN TO SEE THE RESULTS OF ALL CHECK-EXPECTS AND
... DISPLAY STATEMENTS FOR PROBLEMS 1 THROUGH 4 IN THE IW.
... A LINE AT THE BOTTOM OF THE IW SHOULD SAY: ALL 11 TESTS
... PASSED!