; Statement Forms — except step, which is described later in its own section

; Variable Definition: (define variable-name expression)
; — evaluates the expression, and defines variable-name to refer to its value

; Function Definition: (define (function-name parameter-name etc) body-expression)
; — the part ... (function-name parameter-name etc) ... is known as the "header"
; — defines function-name as a new literal referring to a function that can be called with as many argument values
; as the number of parameters
; — when called, the literal argument values are substituted into the body in place of the corresponding parameter names

; Assertion / Claim / Test ...
(same! expression expression)
; — evaluates the expressions, compares them with same?, and reports a problem if the two expressions produced different values
(true! condition-expression) ; ... is equivalent to ... (same! condition-expression #true)
(false! condition-expression) ; ... is equivalent to ... (same! condition-expression #false)

; Show Function Anonymously: (anonymous function-name)
; — for a function name from a function definition, show its equivalent anonymous function literal

; Expression Forms

; Literal Value ...
; • Number, in decimal or fractional form: ±n.n ±n/n
; • Image, pasted into the code:
; • Text, as a sequence of characters between double-quotes: "...characters...",
; — in particular ... "" ... is known as the "empty text"
; • Boolean, as one of two special keywords to represent truth or falsity: #true #false
; • List, as a parenthesized grouping of the name list and its literal elements: (list literal-value etc)
; — in particular ... (list) ... is known as the "empty list"
; • Function ...
; ... as one of the function names from our language or from a function definition: function-name
; ... as an "anonymous" grouping of a header and body, with fun in place of the function name: ((fun parameter-name etc) body-expression)

; Note: for detailed meanings of the following non-literal expressions, see the description of Stepping in the Statement Forms.

; Variable Reference: variable-name ; — one of the variable names from our language or from a variable definition
; — produces the value of the variable

; Function Call: (function-expression argument-expression etc)
; — evaluates each argument expression to produce a literal value, then "calls" the function giving it those values
; Conditionals ...

(if condition-expression consequent-expression
  condition-expression consequent-expression
  etc
  else alternative-expression)
; — evaluates the conditions in order until the first one that produces #true, then evaluates the corresponding consequent,
; or if none produce #true then it evaluates the alternative

(and condition-expression condition-expression etc)
; — evaluates the conditions one at a time, “delaying evaluation” of later conditions, to determine whether they are all #true
; — stops early (“short circuits”), if one of them is #false

(or condition-expression condition-expression etc)
; — evaluates the conditions one at a time, “delaying evaluation” of later conditions, to determine whether at least one is #true
; — stops early (“short circuits”), if one of them is #true

; Show Evaluation Steps
; ================
; You may always show any skipped steps: we use hide options in the exam to save you the time/work of writing out some of the steps,
; In particular you may show any steps to evaluate any expression we ask you about, even if we do not ask only for its value.

(step expression)
(step (hide hide-option etc)
  expression)

; A hide-option is one of ...

(function-name literal-value etc) ; — skip showing body substitution and evaluation of that call if it appears

function-name ; — skip showing body substitution and evaluation of any call of that function, except if it's in the original expression

if-conditions ; — skip showing the steps to evaluate the condition(s), and choose the consequent/alternative next

if-introduction ; — skip the substitution step for bodies that are if expressions, and choose the consequent/alternative instead

; Stepping a Function Call Expression ...

(function-literal literal-value etc) ; except when in the form of a literal list

; Defined or anonymous function: copy the body, replace the parameters with the literal arguments
; — but skip to the final value if the function call is a hide option,
; — or skip to the final value if the function name is a hide option and this is part of the original expression,
; — but if number of arguments doesn't match number of parameters then this is an error

; Map / combine: match the left side of its rules shown below, then rewrite it to the form on the right side
; — but if the function call does not match its pattern this is an error

(map f (list a b c etc) \rightarrow (list (f a) (f b) (f c) etc)
(combine f (list a b c etc)) \rightarrow (f a b c etc)

; Any other function from our language: substitute the directly computed value (see the later section describing our lanugage's functions)
; — but if wrong number or type of arguments for that function this an error
; Stepping a Conditional Expression …

; We will only ask for steps for or / and with two conditions, and steps for if with one condition.

; If the underlined condition shown here is not literal then step it …

  (and condition condition)  (or condition condition)
  (and #true condition)      (or #false condition)

; … otherwise, if that condition is a literal boolean then produce that boolean …

   (and #false …) → #false  (or #true …) → #true
   (and #true #true) → #true  (or #false #false) → #false
   (and #true #false) → #false  (or #false #true) → #true

; … otherwise, that condition is a non-boolean literal value, so this is an error.

; If the underlined condition shown here is not literal then step it …

  (if condition …) ; but skip to the consequent/alternative when if-conditions is a hide option

; … otherwise, choose the consequent or alternative accordingly …

  (if #true consequent else alternative) → consequent
  (if #false consequent else alternative) → alternative

; … otherwise, that condition is a non-boolean literal value, so this is an error.

; Stepping a Variable Reference:

variable-name → value  ; substitute the variable’s previously-computed literal value

; Function Design Forms

;==================

; ● Goal Example  (same! (function-name argument etc)
      literal-expression)

; ● Full Design    (same! (function-name argument etc)
      fully-generalizable-expression)

; — when the generalizable expression uses the arguments explicitly, and exactly as-is, so can be generalized in a definition

; — in particular, the first step of …  (step (function-name argument etc)) … would produce the expression

; ● Partial Design  (same! (function-name argument etc)
      expression)

; — when the expression is not literal nor fully generalizable
: Functions in Our Language
:

*Equality Predicate:* | (true! (same? (+ 1 1) 2)) | (false! (same? 3 2))

*Boolean Negation:* | (false! (not #true)) | (true! (not #false))

*Type Predicates:* | (true! (function? flip)) | (true! (list? (list "z" ▲)))
(true! (boolean? #false)) | (true! (number? -12))
(true! (text? "Hi!")) | (true! (image? ☀))

*Function Predicates:* | (true! (unary? flip)) | (false! (binary? flip))

: ● List Functions ●

(define L (list ▲ #true 5 "z")); example list used below ...

(same! (list (filled-triangle 9) (zero? 0) (+ 2 3) "z") L)
(same! (join (list ▲ #true) (list 5 "z")) L)
(true! (empty? (list))) | (same! (length L) 4)
(same! (first L) ▲) | (same! (third L) 5)
(same! (second L) #true) | (same! (fourth L) "z")

(same! (rest L) (list (list #true 5 "z"))
(same! (reverse L) (list "z" 5 #true ▲))
(same! (select positive? (list 0 -1 2 -3 4)) (list 2 4))
(same! (range 4) (list 0 1 2 3)) | (same! (range 2 5) (list 2 3 4))

: ● Image Functions ●

(same! (triangle 9 ▲) | (same! (filled-triangle 9) ▲)
(same! (circle 9 ○) | (same! (filled-circle 9) ○)
(same! (square 9 □) | (same! (filled-square 9) □)
(same! (oval 9 15) ○) | (same! (filled-oval 7 15) ○)
(same! (rectangle 9 15) □ | (same! (filled-rectangle 7 15) □)

(same! (width (oval 9 15)) 9)
(same! (height (oval 9 15)) 15)

(same! (mirror ←) →)
(same! (flip ↓) ↑)
(same! (rotate → 30))
(same! (clockwise ←) ↓)
(same! (anti-clockwise →) ↑)
(same! (scale ▲ 1.5) ▲)
(same! (shrink ▲ ▲)
(same! (enlarge ▲ ▲)
(same! (scale-width 1.5))
(same! (wider ))
(same! (thinner ))
(same! (scale-height 1.5))
(same! (taller ))
(same! (shorter ))

(same! (above )
(same! (align-lefts )
(same! (align-rights )
(same! (beside )
(same! (align-tops )
(same! (align-bottoms )

: • Numeric Functions •

(same! (+ 2 10 3) 15) (same! (- 12) -12) (same! (inc 20) 21)
(same! (* 2 10 3) 60) (same! (- 12 3) 9) (same! (dec 20) 19)
(same! (/ 12 3) 4)

(true! (zero? 0)) (false! (positive? -3))
(true! (positive? 7)) (false! (positive? 0))

(same! (minimum 7 3 9 5) 3)
(same! (maximum 7 3 9 5) 9)

(true! (= 2 2))
(false! (= 2 1))

(true! (< 1 2 3))
(false! (< 1 2 2 3))

(true! (≤ 1 2 2 3))
(true! (≥ 3 2 2 1))

(false! (≥ 3 1 2))
(true! (≥ 3 2 1))

(true! (even? 4)) (false! (odd? 4))
(false! (even? 5)) (true! (odd? 5))

: • Text Functions •

(same! (text-length "one") 3)
(same! (text-join "Hi" " human" "!" "Hi human!"))
(same! (sub-text "abcdefgh" 2 5) "cde")

(same! (text->image "Hi!") Hi)
(same! (text->list "Hi!") (list "H" "i" "!"))
(same! (number->text 104) "104")