; A program is a sequence of “top level” expressions and statements.

; ● Expression Forms ●

; Literal Value

```
▣ ; inserted/pasted image
function-name ; function by name, from the language or from a definition
±n.n ±n/n ; number as decimal or fraction
#true #false ; boolean
"⋯characters⋯" ; text
(list literal-value etc) ; list
```

; Variable Reference

```
variable-name ; variable, from a definition
```

; Function Call

```
(function-name expression etc)
; — the expressions after the function name are the “argument” expressions
```

; ● Statement Forms ●

; Definition of Variable

```
(define variable-name expression)
; — the expression after the variable name is the “value” expression
```

; Definition of Function

```
(define (function-name parameter-name etc)
  expression)
; — the parenthesized function name with parameter names is the “header”
; — the expression after the header is the “body” expression
```

; Assertion / Test

```
(same! expression
       expression
       etc)
```

; Reveal Algebraic Evaluation Sequence

```
(step expression)
```

; Show the sequence of expressions produced by replacing sub-expressions
; that are in the following forms, until that produces the literal value of the
; expression or reports an error and stops.

```
… (function-name literal-value etc) …
```

; • For the function map or combine : match its first pattern below to determine
; the literals f a b c ... , then substitute those literals into its rule's second pattern.
; If the expression doesn't match its pattern report an error.

```
(map  f  (list      a          b          c    etc))

 →       (list ( f  a ) ( f  b ) ( f  c ) etc)

(combine  f  (list  a  b  c  etc))

 →            ( f   a  b  c  etc)
```

; • For a function (other than map or list) from our language: substitute a directly
; computed value, or report an error if there are the wrong number or kind of
; arguments needed by the function.

; • For a function from a definition: copy its body and substitute the arguments
; in place of the parameter names wherever those names occur in the body,
; or report an error if the number of arguments and parameter names differ.

```
… variable-name …
 → literal-value
```

; Function Examples

; ● Type Predicates ●

```
(same? (image? ★ )      #true)  | (same! (boolean? #false)   #true)
(same! (function? flip) #true)  | (same! (text? "Hi!")       #true)
(same! (number? -12.3)  #true)  | (same! (list? (list ★ 5)) #true)
```

; ● Function Predicates ●

```
(same! (unary? flip) #true) | (same! (binary? flip) #false)
```

; ● Image Functions ●

```
(same! (mirror ←) →)                 (same! (scale-width  1.5)
(same! (flip ↓) ↑)                      )
(same! (turn → 30) ↘)                (same! (scale-height  1.5)
(same! (clockwise → ) ↓)                )
(same! (anti-clockwise → ) ↑)        (same! (wide ) )
(same! (scale △ 1.5) △)              (same! (thin ) )
(same! (small △) △)                  (same! (tall ) )
(same! (large △) △)                  (same! (short ) )

(same! (above ● ★ △) ●★△)            (same! (triangle 9) △)
(same! (above-left ● ★ △) ●★△)       (same! (circle   9) ○)
(same! (above-right ● ★ △) ●★△)      (same! (square   9) □)
                                     (same! (oval     9 15) ⬯)
                                     (same! (rectangle 9 15) ▯)

(same! (beside ● ▲ ✶)                (same! (solid-triangle 9) ▲)
  ●▲✶)                               (same! (solid-circle   9) ●)
(same! (beside-top ● ▲ ✶)            (same! (solid-square   9) ■)
  ●▲✶)                               (same! (solid-oval      7 15) ⬮)
(same! (beside-bottom ● ▲ ✶)         (same! (solid-rectangle 7 15) ▮)
  ●▲✶)

(same! (width (oval 7 15)) 7) | (same! (height (oval 7 15)) 15)
```

; ● Numeric Functions ●

```
(same! (+ 2 10 3) 15) | (same! (- 12)  -12) | (same! (/ 12 3) 4)
(same! (* 2 10 3) 60) | (same! (- 12 3)  9)
```

; ● Text Functions ●

```
(same! (text-length "one") 3)
(same! (text->image "Hi!") Hi!)
(same! (text-join "Hi" " human!") "Hi human!")
```

; ● List Functions ●

```
(same! (list (star 10) (+ 2 3) (text? 4)) (list ★ 5 #false))

(same! (length (list ★ 5 #false)) 3)

(same! (range   8) (list 0 1 2 3 4 5 6 7))
(same! (range 3 8)       (list 3 4 5 6 7))
(same! (range 3 8 2)     (list 3   5   7))
```