; A program is a sequence of 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
```

; 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 stops and reports an error.

... (function-name literal-value etc) ...

; • For the function map : match the 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)

; • 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

; Substitute the value that was computed when the variable was defined.
```

; Function Examples

; • Type Predicates •

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

; • Function Predicates •

```
(unary?  flip)  #true     (binary? flip)  #false
```

; • Image Functions •

```
(mirror ★)
(flip ★)
(turn ★ 30)
(clockwise ★)
(anti-clockwise ★)
(scale △ 1.5)
(small △)
(large △)
(scale-width 🐟 1.5)
(scale-height 🐟 1.5)
(tall 🐟)
(short 🐟)
(thin 🐟)
(wide 🐟)
(triangle 9)        (solid-triangle 9)
(circle   9)        (solid-circle   9)
(square   9)        (solid-square   9)
(oval 9 15)         (solid-oval     9 15)
(rectangle 9 15)    (solid-rectangle 9 15)

(width  (oval 9 15))   9     (height (oval 9 15))  15

(above-left ● ★ △)
(above ● ★ △)
(above-right ● ★ △)
(beside-top ● ▲ ★)
(beside ● ▲ ★)
(beside-bottom ● ▲ ★)
```

; • Numeric Functions •

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

; • Text Functions •

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

; • List Functions •

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

(length (list ★ 5 #false))  3
```