; 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` ; from a definition

; • Function Call : `(function-name argument-expression etc)`

; • Parameter Reference : `parameter-name` ; in the body of a function definition

; • Conditional :
```
(if condition-expression consequent-expression
    else alternative-expression)
```

; ● Statement Forms ●

; • Definition of Variable or Function •
```
(define variable-name value-expression)
(define (function-name parameter-name etc) ; “header”
  body-expression)
```

; • Assertion / Test :
```
(same! expression
       expression
       etc)
```
```
(true!  expression)
(false! expression)
```

; • Inspect Evaluation •

```
(step expression)
```
```
(step (hide function-name/call etc)
      expression)
```

; Show expressions produced by replacing sub-expressions that are in the following forms,
; until reaching the literal value of the expression or encountering an error.

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

; • For a function from a definition :
;  If the number of arguments and parameter names differ : report an error.
;  If the function name or this whole call is a hide option : skip to result value.
;  Otherwise : copy the function's body and substitute the arguments in place of
;  the parameter names wherever those names occur in the body.
; • For the function map or combine : match its first pattern below, then :
;  If the expression doesn't match its pattern : report an error.
;  Otherwise : determine the literal values for f a b c ... , then substitute those
;  into the rule's second pattern.

```
(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 any other function from our language :
;  If there are the wrong number or kind of arguments : report an error.
;  Otherwise : substitute a directly computed value (see the “Function Examples”)

```
variable-name → literal  ; • Substitute variable's previously-computed value.
```

```
(if ···)  ; • Based on the evaluation state of the condition :
```
```
(if #true consequent
    else alternative)
  → consequent
```
```
(if #false consequent
    else alternative)
  → alternative
```

```
(if non-boolean-literal ···)  ; report an error
```

```
(if condition ···)  ; evaluate condition first
```

; ● Function Design ●

; Goal Example : `(same! (function-name argument etc) literal)`

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

; ... where the generalizable expression only uses the arguments as-is, so it can be used
; as the body of the function's definition by replacing arguments with parameter names.

; Partial Design :
```
(same! (function-name argument etc)
       partially-general-expression)
```

; ... where the partially general expression is not fully generalizable, but not just literal.

; Function Examples

; ● Equality Predicate : `(true! (same? (+ 1 1) 2))` | `(false! (same? 3 2))`

; ● Type Predicates ●

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

; ● Function Predicates : `(true! (unary? flip))` | `(false! (binary? flip))`

; ● Image Functions ●

```
(same! (mirror  )
(same! (flip  )
(same! (turn  30)  )
(same! (clockwise  )  )
(same! (anti-clockwise  )  )
(same! (scale  1.5)  )
(same! (small  )  )
(same! (large  )  )
```
```
(same! (scale-width  1.5)
        )
(same! (scale-height  1.5)
        )
(same! (wide  )  )
(same! (thin  )  )
(same! (tall  )  )
(same! (short  )  )
```

```
(same! (above  )  )
(same! (above-left  )  )
(same! (above-right  )  )
```
```
(same! (triangle 9)  )
(same! (circle   9)  )
(same! (square   9)  )
(same! (oval     7 15)  )
(same! (rectangle 7 15)  )
```

```
(same! (beside  )
        )
(same! (beside-top  )
        )
(same! (beside-bottom  )
        )
```
```
(same! (solid-triangle 9)  )
(same! (solid-circle   9)  )
(same! (solid-square   9)  )
(same! (solid-oval     7 15)  )
(same! (solid-rectangle 7 15)  )
```

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

; ● Numeric Functions ●

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

```
(same! (number->text -12) "-12")
```

; ● Text Functions ●

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

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

; ● List Functions ●

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

```
(same! (length  (list  5 #false)) 3)
```

```
(same! (first   (list  5 #false))  )
(same! (rest    (list  5 #false)) (list 5 #false))
```

```
(same! (reverse (list  5 #false)) (list #false 5  ))
```

```
(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))
```