A program is a sequence of expressions and statements.

Convention: "condition" stands for an expression intended to evaluate to a boolean.

* Expression Forms *

### Literal Value

- Function: by name from a definition or our language, or created anonymously.
  
  ```
  function-name ((fun parameter-name etc) "header") body-expression
  ```

- If true/false: boolean "...characters"; test

- Variable Reference: variable-name from a definition or our language

- Function Call: (function-name argument-expression etc)

- Conditional: (if condition consequent-expression else alternative-expression)

* Statement Forms *

- User Definition of a Variable or Function
  
  ```
  (define variable-name value-expression)
  ```

- Assert/Test: (same expression expression) (true! condition) (false! condition)

* Algebraic Evaluation Inspection*

```step expression

(step (hide (function-name literal-value etc)) expression)
```

- (function-literal literal-value etc) ... : except literal list
- : defined or anonymous function: copy body, replace parameters with arguments
- : except if it matches the function call after hide, then skip to final value
- : if number of arguments doesn’t match number of parameters: an error
- : combine / map: match as rule’s pattern, rewrite as rule’s template
- : if expression does not match its pattern: 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)
```

- any other function from our language: substitute directly computed value
- : if wrong number or type of arguments for that function: an error

```... (if true consequent-expression
else alternative-expression) ...
  ⇒ consequent-expression
... (if false consequent-expression
else alternative-expression) ...
  ⇒ alternative-expression
... (if non-boolean-literal ...) ... ; an error
... (if condition-expression ...) ... ; evaluate condition first
```

- variable-name ... ⇒ literal-value
  
  Substitute the value that was computed when the variable was defined.

* Equality Predicate:*

```(true! (same? (+ 1 1) 2)) (false! (same? 3 2))```

* Type Predicates:*

```(true! (function? flip)) (true! (list? (list "z" "a")))
(true! (boolean? #false)) (true! (number? -12))
(true! (text? "Hi")) (true! (image? "hi"))```

* Function Predicates:*

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

* List Functions:*

```(same! (list (filled-triangle 9) (zero? 0) (+ 2 3) "hi")
(list #true 5 "hi"))
(same! (map - (list 3 1 7) (list -3 -1 -7))
(same! (combine + (list 3 1 7)) 11)
(true! (empty? (list)))
(same! (length (list #true 5 "hi")) 4)
(same! (first (list #true 5 "hi")) #true)
(same! (second (list #true 5 "hi")) 5)
(same! (fourth (list #true 5 "hi")) "hi")
(same! (rest (list #true 5 "hi")) (list #true 5 "hi"))
(same! (reverse (list #true 5 "hi")) (list "hi" #true #true #true #true))
(same! (range 4) (list 0 1 2 3))```

* Image Functions:*

```(same! (mirror #true 5 "hi"))
(same! (scale-width 1.5 #true 5 "hi"))
(same! (wider #true 5 "hi"))
(same! (thinner #true 5 "hi"))
(same! (scale-height 1.5 #true 5 "hi"))
```

* Numeric Functions:*

```(same! (+ 2 10 3) 15)
(same! (inc 20) 21)
(same! (dec 20) 19)
```

```(true! (zero? 0)) (false! (positive? -3))
true! (positive? 7) false! (positive? 0)
true! (false? (= 3 3)) false! (false? (= 3 2 3))```

* Text Functions:*

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