Exceptions are raised with a raise statement.

\[ \text{raise <expr>} \]

<expr> must evaluate to a subclass of BaseException or an instance of one.

\[ \text{try:} \]
\[ \text{except <exception class> as <name>:} \]
\[ \text{except <suite> } \]

The <try suite> is executed first. If, during the course of executing the <try suite>, an exception is raised that is not handled otherwise, and if the class of the exception inherits from <exception class>, then the <except suite> is executed, with <name> bound to the exception.

(define (car ones) -> 1)
(define (cdr ones) -> nil)
(define (cons-stream a b) -> (a . b))
(define (null? l) -> (eq? l nil))
(define (pair? x) -> (and (null? (car x)) (null? (cdr x))))

(define (map-stream f s) -> (f (car s)) map-stream (f (cdr s))))

(define (cons-stream a b) -> (a . b))
(define (null? l) -> (eq? l nil))
(define (pair? x) -> (and (null? (car x)) (null? (cdr x))))

The built-in Scheme list data structure can represent combinations
scm (list 'quote 10 2) scm> (eval (list 'quote 10 2))
(quote 10 2) 5
A macro is an operation performed on source code before evaluation
(define-macro (twice expr) -> > (twice (print 2)))
(define-macro (begin expr expr) -> 2)
Evaluation procedure of a macro call expression:
- Evaluate the operand sub-expression, which evaluates to a macro
- Call the macro procedure on the operand expressions
- Evaluate the expression returned from the macro procedure

A procedure call that has not yet returned is active. Some procedure calls are tail calls. A Scheme interpreter should support an unbounded number of active tail calls.

A tail call is a call expression in a tail context, which are: The last body expression in a lambda expression
Expressions 2 & 3 (consequent & alternative) in a tail context if
All non-predicate sub-expressions in a tail context cond
The last sub-expression in a tail context and, or, begin, or let

(define (factorial n k) -> \[ \text{Not a tail call} \]
(define (length tail s) -> Recursive call is a tail call
(define (length-iter s n) -> if (null? s), n =
(length-iter (cdr s) (+ 1 n)))
(length-iter s 0))

(define (factorial n k) -> \[ \text{Not a tail call} \]
(define (length s) -> \[ \text{Not a tail call} \]
(if (= n 0) k
(\text{Factorial} \ (- n 1))
(+ 1 (length (cdr s)))
(length (cdr s) 0)

\[ \text{Dynamic scope: The parent of a frame is the environment in which a procedure was defined. (lambda ...)} \]
\[ \text{Lexical scope: The parent of a frame is the environment in which a procedure was defined. (lambda ...)} \]
A basic interpreter has two parts: a parser and an evaluator.

Each element can be a combination or atom (primitive).

A script takes a sequence of lines and returns an expression.

The task of parsing a language involves coercing a string representation of an expression into the expression itself. Parsers must validate that expressions are well-formed.

A parser takes a sequence of lines and returns an expression.

In the late 1950s, computer scientists used confusing names.

Each element is either a combination or atom (primitive).

A Parser takes a sequence of lines and returns an expression.

A Scheme list is written as elements in parentheses.

A Scheme list is written as elements in parentheses.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

Each element can be a combination or atom (primitive).

A Scheme list is written as elements in parentheses.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.

A list is an arbitrary value for the second element of the last pair. Dotted lists may not be well-formed.