Exceptions are raised with a raise statement.

try <try suite> except <exception class> as <name>: <except suite>

The <except 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.

Recursive decomposition:

- Finding simpler instances of a problem.
  - E.g., count_partitions(6, 4)
  - Explore two possibilities:
    - Use at least one 4
    - Don’t use any 4
  - Solve two simpler problems:
    - count_partitions(2, 4)
    - count_partitions(6, 3)
  - Tree recursion often involves exploring different choices.

<table>
<thead>
<tr>
<th>A stream is a Scheme pair, but the cdr is evaluated lazily</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>A built-in Scheme list data structure can represent combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(define-macro (twice expr) (&gt; (twice (print 2)) 2) )</td>
</tr>
<tr>
<td>(define-macro (twice expr) (&gt; (twice (print 2)) 2) )</td>
</tr>
<tr>
<td>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:</td>
</tr>
<tr>
<td>- The last body expression in a tail context</td>
</tr>
<tr>
<td>- Expressions 2 &amp; 3 (consequent &amp; alternative) in a tail context</td>
</tr>
<tr>
<td>- All non-predicate sub-expressions in a tail context</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A column has a name and a type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A table has columns and rows</td>
</tr>
<tr>
<td>--------------------------------</td>
</tr>
<tr>
<td>SELECT [expression] AS [name], [expression] AS [name], ...;</td>
</tr>
<tr>
<td>SELECT [columns] FROM [table] WHERE [condition] ORDER BY [order];</td>
</tr>
<tr>
<td>CREATE TABLE parents AS</td>
</tr>
<tr>
<td>SELECT &quot;abraham&quot; AS parent, &quot;barack&quot; AS child UNION</td>
</tr>
</tbody>
</table>
| SELECT "abraham" AS name, "long" AS for UN;
| SELECT "barack" AS name, "short" AS for UNION |
| SELECT "clinton" AS name, "long" AS for UNION |
| SELECT "delano" AS name, "short" AS for UNION |
| SELECT "eisenhower" AS name, "short" AS for UNION |

The number of groups is the number of unique values of an expression having count(+)=1; weight/legs group by weight/legs, count(*) from animals; having count(+)=1; weight/legs;

The way in which names are looked up in Scheme and Python is called lexical scope (or static scope).

Lexical scope: The parent of a frame is the environment in which a procedure was defined. (lambda ...)

Dynamic scope: The parent of a frame is the environment in which a procedure was called. (mu ...)
Scheme programs consist of expressions, which can be:

- **Primitive expressions:** 2 3.1 true # quotient ...
- **Combinations:** (quotient 10 2), (not true)

Numbers are self-evaluating; symbols are bound to values.

Call expressions have an operator and 0 or more operands.

A combination that is not a call expression is a special form:

- **If expression:** (if predicate <consequent> <alternative>)
- **Define naming:** (define <name> <expression>)
- **New procedures:** (define (<name> <formal parameters>) <body>)

Two equivalent expressions:

- (define (plus4 x) (+ x 4))
- (define plus4 (lambda (x) (+ x 4)))

An operator can be a combination too:

- (((lambda (x y z) (+ x y (square z))) 1 2 3))
- (define (plus4 x) (+ x 4))
- (define plus4 (lambda (x) (+ x 4)))

A Scheme list is written as space-separated combinations.

- (define (cons 1 2))
- (define cons (lambda (x y) (list x y)))

A number or a Pair with an operator as its first element

A Scheme list is written as elements in parentheses.

- (pair 1 2)
- (define (pair 1 2))
- (define pair (lambda (x y) (list x y)))

Numbers are self-evaluating;

- (define (square x) (* x x))
- (square 3)

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

- (+ 3 (+ 2 4) (+ 3 5))

The task of parsing a language involves coercing a string representation of an expression into a representation itself.

Parsers must validate that expressions are well-formed.

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

- (define (f s) (if null? s nil s))
- (f (list 1 2))

A basic interpreter has two parts: a parser and an evaluator.

- (define (eval operator operands)
  (cond ((null? operands) nil)
        (else ( operator (eval (car operands)) (eval (cdr operands)) ) )))

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

- (+ 3 (+ 2 4) (+ 3 5))

The task of parsing a language involves coercing a string representation of an expression into a representation itself.

Parsers must validate that expressions are well-formed.

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

- (define (eval operator operands)
  (cond ((null? operands) nil)
        (else ( operator (eval (car operands)) (eval (cdr operands)) ) )))

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

- (+ 3 (+ 2 4) (+ 3 5))

The task of parsing a language involves coercing a string representation of an expression into a representation itself.

Parsers must validate that expressions are well-formed.

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

- (define (eval operator operands)
  (cond ((null? operands) nil)
        (else ( operator (eval (car operands)) (eval (cdr operands)) ) )))

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

- (+ 3 (+ 2 4) (+ 3 5))

The task of parsing a language involves coercing a string representation of an expression into a representation itself.

Parsers must validate that expressions are well-formed.

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

- (define (eval operator operands)
  (cond ((null? operands) nil)
        (else ( operator (eval (car operands)) (eval (cdr operands)) ) )))

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

- (+ 3 (+ 2 4) (+ 3 5))

The task of parsing a language involves coercing a string representation of an expression into a representation itself.

Parsers must validate that expressions are well-formed.