The exception suite is executed, with name bound to the exception. If the class of the exception inherits from exception class, then the exception suite is executed with name bound to the exception.

The try statement 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...
Scheme programs consist of expressions, which can be:
- Primitive expressions: 2 3.3 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 bindings: (define <name> <expression>)
- New procedures: (define (= <name> <formal parameters>) <body>)

Lambda expressions evaluate to anonymous procedures.
(lambda (= <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)

In the late 1950s, computer scientists used confusing names.
- cons: Two-argument procedure that creates a pair.
- car: Procedure that returns the first element of a pair.
- cdr: Procedure that returns the second element of a pair.
- nil: The empty list.

They also used a non-obvious notation for linked lists.
- A (linked) Scheme list is a pair in which the second element is nil or a Scheme list.
- Scheme lists are written as space-separated combinations.

Expressions are well-formed if:
- They form a Scheme list.
- They have no nil elements.
- They are not ill-formed.

A Scheme list is written as elements in parentheses:
(Pair( <element> ) ... <element> )

Expression Trees

Symbol normally refer to values; how do we refer to symbols?
- Symbols normally refer to values; how do we refer to symbols?

Syntactic analysis identifies the hierarchical structure of an expression, which may be needed.
Each call to scheme_read consumes the input tokens for exactly one expression.

Base case: symbols and numbers

Recursive call: scheme_read sub-expressions and combine them

Lambda Procedure instance [parent=g]

How to design functions:
1) Identify the information that must be represented and how it is represented. Illustrate with examples.
2) State what kind of data the desired function consumes and produces. Formulate a concise answer to the question what the function computes.
3) Work through examples that illustrate the function's purpose.
4) Outline the function as a template.
5) Fill in the gaps of the function template. Exploit the purpose statement and the examples.
6) Convert examples into tests and ensure that the function passes them.