Scheme is a Dialect of Lisp

What are people saying about Lisp?

• "You don’t know Lisp, you don’t know what it means for a programming language to be powerful and elegant."
  - Richard Stallman, created Emacs & the first free variant of UNIX
• "The only computer language that is beautiful."
  - Neal Stephenson, DeMero’s favorite sci-fi author
• "The greatest single programming language ever designed."
  - Alan Kay, co-inventor of Smalltalk and GD (from the user interface video)

Scheme Expressions

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 include an operator and 0 or more operands in parentheses

> (quotient 10 2)
5
> (quotient (+ 8 7) 5)
3

"quotient" names Scheme’s built-in integer division procedure (i.e., function)
Combinations can span multiple lines (spacing doesn’t matter)

Special Forms

A combination that is not a call expression is a special form:
• if expression: (if <predicate> <consequent> <alternative>)
• and or: (and <e1> ... <en>), (or <e1> ... <en>)
• Binding symbols: (define <symbol> <expression>)
• New procedures: (define (<symbol> <formal parameters>) <body>)

> (define pi 3.14)
> (* pi 2)
6.28
> (define (abs x)
  (if (< x 0)
    (- x)
    x))
> (abs -3)
3

The symbol "pi" is bound to 3.14 in the global frame
A procedure is created and bound to the symbol "abs"

Scheme Interpreters
Lambda Expressions

Lambda expressions evaluate to anonymous procedures

\[ \lambda (\text{formal-parameters}) \text{body} \]

Two equivalent expressions:

\[
\text{(define (plus4 x) (+ x 4))} \\
\text{(define plus4 (lambda (x) (+ x 4)))}
\]

An operator can be a call expression too:

\[
\text{((lambda (x y z) (+ x y (square z))) 1 2 3)}
\]

Evaluates to the \[ x+y+z \]

Sierpinski's Triangle

More Special Forms

Cond & Begin

The cond special form that behaves like if-elif-else statements in Python

\[
\text{if } x > 10: \\
\text{\quad print('big')} \\
\text{elif } x > 5: \\
\text{\quad print('medium')} \\
\text{else:} \\
\text{\quad print('small')} \\
\]

The begin special form combines multiple expressions into one expression

\[
\text{if } x > 10: \\
\text{\quad (begin (print 'big) (print 'guy))} \\
\text{else:} \\
\text{\quad (begin (print 'small) (print 'fry))} \\
\]

Let Expressions

The let special form binds symbols to values temporarily; just for one expression

\[
\text{a = 3} \\
\text{b = 2 + 2} \\
\text{c = math.sqrt(a * a + b * b)} \\
\text{(define c (let ((a 3) (b (+ 2 2))) (sqrt (+ (* a a) (* b b)))))}
\]

Lists

Important! Scheme lists are written in parentheses with elements separated by spaces

\[
> (\text{cons 1 (\text{cons 2 nil))}) \\
(1 2) \\
> (\text{define x (cons 1 (cons 2 nil))}) \\
> (x) \\
(1) \\
> (\text{car x}) \\
1 \\
> (\text{cdr x}) \\
(2)
\]

Scheme Lists

In the late 1950s, computer scientists used confusing names

- \text{cons}: Two-argument procedure that creates a linked list
- \text{car}: Procedure that returns the first element of a list
- \text{cdr}: Procedure that returns the rest of a list
- \text{nil}: The empty list

Important! Scheme lists are written in parentheses with elements separated by spaces
Symbolic Programming

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

> (define a 1)
> (define b 2)
> (list a b)
(1 2)

Quotation is used to refer to symbols directly in Lisp.

> (list 'a 'b)
(a b)
> (list 'a b)
(a 2)

Short for (quote a), (quote b): Special form to indicate that the expression itself is the value.

Programs as Data

A Scheme Expression is a Scheme List

Scheme programs consist of expressions, which can be:

• Primitive expressions: 2 3.3 true + quotient
• Combinations: (quotient 10 2) (not true)

The built-in Scheme list data structure (which is a linked list) can represent combinations

(scm> (list 'quotient 10 2))
(quotient 10 2)
(scm> (eval (list 'quotient 10 2)))
5

In such a language, it is straightforward to write a program that writes a program

(Demo)

Generating Code

Quasiquotation

There are two ways to quote an expression

Quote: '(a b) => (a b)
Quasiquote: `(a b) => (a b)

They are different because parts of a quasiquoted expression can be unquoted with ,
(define b 4)
Quote: '(a ,(+ b 1)) => (a (unquote (+ b 1))
Quasiquote: `(a ,(+ b 1)) => (a 5)

Quasiquotation is particularly convenient for generating Scheme expressions:
(define (make-add-procedure n) `(lambda (d) (+ d ,n)))
(make-add-procedure 2) => (lambda (d) (+ d 2))

Example: While Statements

What's the sum of the squares of even numbers less than 10, starting with 2?

(x = 2
 total = 0
while x < 10:
total = total + x * x
x = x + 2)

(define (f x total)
(if (< x 10)
(f (+ x 2) (+ total (* x x)))
total))
(begin (f 2 0))

What's the sum of the numbers whose squares are less than 50, starting with 1?

(x = 1
 total = 0
while x * x < 50:
total = total + x
x = x + 1)

(define (f x total)
(if (> (* x x) 50)
(f (+ x 1) (+ total x))
total))
(begin (f 1 0))

(Demo)