Scheme is a Dialect of Lisp

What are people saying about Lisp?

- “If 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, DeNero's favorite sci-fi author

- “The greatest single programming language ever designed.”
  - Alan Kay, co-inventor of Smalltalk and OOP (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

```scheme
> (quotient 10 2)
5
> (quotient (+ 8 7) 5)
3
> (+ (* 3 (+ (* 2 4) (+ 3 5))) (+ (- 10 7) 6))
```

“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>)

```scheme
> (define pi 3.14)
> (* pi 2)
6.28
> (define (abs x)
    (if (> x 0)
        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

Evaluation:
1. Evaluate the predicate expression
2. Evaluate either the consequent or alternative
Lambda Expressions

Lambda expressions evaluate to anonymous procedures

\[
\lambda \langle \text{formal-parameters} \rangle \langle \text{body} \rangle
\]

Two equivalent expressions:

\[
\begin{align*}
\text{(define (plus4 x) (+ x 4))} \\
\text{(define plus4 (lambda (x) (+ x 4)))}
\end{align*}
\]

An operator can be a call expression too:

\[
\text{((lambda (x y z) (+ x y (square z))) 1 2 3)} \rightarrow 12
\]

Evaluates to the \text{x+y+z} procedure

More Special Forms

Cond & Begin

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

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

The begin special form combines multiple expressions into one expression

\[
\begin{align*}
\text{if } x > 10: & \quad \text{(print 'big')} \\
\text{elif } x > 5: & \quad \text{(print 'medium')} \\
\text{else:} & \quad \text{(print 'small')} \\
\end{align*}
\]
Let Expressions

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

```lisp
(define c (let ((a 3) (b (+ 2 2))) (sqrt (+ (* a a) (* b b)))))
```

- `a` and `b` are still bound down here
- `a` and `b` are not bound down here

Turtle Graphics

Drawing Stars

- `(forward 100)` or `(fd 100)` draws a line
- `(right 90)` or `(rt 90)` turns 90 degrees

Sierpinski's Triangle