Announcements

Scheme

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 Fundamentals

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

Special Forms

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

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

Scheme Interpreters

(Demo)
Lambda Expressions

Lambda expressions evaluate to anonymous procedures

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

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 \( x+y+z \) procedure

Pairs and Lists

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) list in Scheme is a pair in which the second element is \( \text{nil} \) or a Scheme list.
- Important! Scheme lists are written in parentheses separated by spaces
- A dotted list has any value for the second element of the last pair; maybe not a list!

```scheme
> (define x (cons 1 2))
> x
(1 . 2)
> (car x)
1
> (cdr x)
2
> (cons 1 (cons 2 (cons 3 (cons 4 nil))))
(1 2 3 4)
```

Symbolic Programming

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

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

Quotation is used to refer to symbols directly in Lisp.

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

Quotation can also be applied to combinations to form lists.

```scheme
> (car '(a b c))
(a)
> (cdr '(a b c))
(b c)
```

Pairs and Lists

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

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

Quotation is used to refer to symbols directly in Lisp.

```scheme
> (list 'a 'b)
(a b)
```

Quotation can also be applied to combinations to form lists.

```scheme
> (car '(a b c))
(a)
> (cdr '(a b c))
(b c)
```

Dots can be used in a quoted list to specify the second element of the final pair.

```scheme
> (cdr (cdr '(1 2 . 3)))
3
```

However, dots appear in the output only of ill-formed lists.

```scheme
> '(1 2 . 3)
(1 2 . 3)
> '(1 2 . (3 4))
(1 2 3 4)
> '(1 2 3 . nil)
(1 2 3 . nil)
> '(1 2 3 . nil)
```

What is the printed result of evaluating this expression?

```scheme
> (cdr '((1 2 . (3 4 . (5)))))
(3 4 5)
```