Monday, March 30
Announcements

• Homework 7 due Wednesday 4/8 @ 11:59pm
• Quiz 3 released Tuesday 4/7, due Thursday 4/9 @ 11:59pm
  ▪ Open note, open interpreter, closed classmates, closed Internet
• Composition corrections for projects 1, 2, & 3 are due Monday 4/13 @ 11:59pm (do them now!)
  ▪ Make changes to your project based on the composition feedback you received
  ▪ Earn back any points you lost on composition
Scheme
Scheme is a Dialect of Lisp

What are people saying about Lisp?

• "The greatest single programming language ever designed."
  – Alan Kay, co-inventor of Smalltalk and OOP (from the user interface video)

• "The only computer language that is beautiful."
  – Neal Stephenson, DeNero's favorite sci-fi author
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

> (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)

(Demo)
Special Forms
A combination that is not a call expression is a special form:

- **if** expression:  \((\text{if } \text{<predicate>} \text{<consequent>} \text{<alternative>})\)
- **and** and **or**:  \((\text{and } \text{<e1> ... <en>}), (\text{or } \text{<e1> ... <en>})\)
- Binding symbols:  \((\text{define } \text{<symbol>} \text{<expression>})\)
- New procedures:  \((\text{define } (\text{<symbol>} \text{<formal parameters>}) \text{<body>})\)

```scheme
> (define pi 3.14)
> (* pi 2)
6.28

> (define (abs x)
  (if (< x 0)
    (- x)
    x))
> (abs -3)
3
```

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

The symbol “pi” is bound to 3.14 in the global frame

A procedure is created and bound to the symbol “abs”
Scheme Interpreters

(Demo)
Lambda Expressions
Lambda Expressions

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 call expression too:

((lambda (x y z) (+ x y (square z))) 1 2 3) 12

Evaluates to the x+y+z^2 procedure
Pairs and Lists
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 **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)
(Demo)
```

Not a well-formed list!
Symbolic Programming
Symbolic Programming

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

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

Quotation is used to refer to symbols directly in Lisp.

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

Quotation can also be applied to combinations to form lists.

```lisp
> (car '(a b c))
a
> (cdr '(a b c))
(b c)
```
Scheme Lists and Quotation

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)
```

What is the printed result of evaluating this expression?

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

(Demo)