

Scheme

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 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)

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

Combinations can span multiple lines (spacing doesn't matter)

(Demo)

Special Forms

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

```
> (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"

(Demo)

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

10

More Special Forms

Cond & Begin

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

```
if x > 10:
    print('big')
elif x > 5:
    print('medium')
else:
    print('small')
```

```
(cond ((> x 10) (print 'big'))
      ((> x 5) (print 'medium'))
      (else (print 'small')))
```

```
(print
 (cond ((> x 10) 'big)
       ((> x 5) 'medium)
       (else 'small))))
```

The begin special form combines multiple expressions into one expression

```
if x > 10:
    print('big')
    print('guy')
else:
    print('small')
    print('fry')
```

```
(cond ((> x 10) (begin (print 'big') (print 'guy')))
      (else (begin (print 'small') (print 'fry'))))
```

```
(if (> x 10) (begin
              (print 'big')
              (print 'guy'))
      (begin
        (print 'small')
        (print 'fry')))
```

12

Let Expressions

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

```
a = 3
b = 2 + 2
c = math.sqrt(a * a + b * b)
```

*a and b are **still** bound down here*

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

*a and b are **not** bound down here*

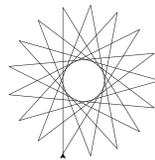
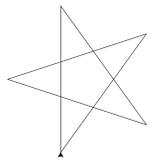
13

Turtle Graphics

Drawing Stars

(forward 100) or (fd 100) draws a line

(right 90) or (rt 90) turns 90 degrees



(Demo)

15

Sierpinski's Triangle

(Demo)