Stream Data Abstraction

```
(cons-stream <x> <y>)  == (cons <x> (delay <y>))

(define (stream-car s) (car s))

(define (stream-cdr s) (force (cdr s)))
```

List HOPs to Stream HOPs

```
(define (list-ref lst n)
  (if (= n 0)
      (car lst)
      (list-ref (cdr lst) (- n 1))))

(define (stream-ref s n)
  (if (= n 0)
      (stream-car s)
      (stream-ref (stream-cdr s) (- n 1))))
```

```
(define (map proc lst)
  (if (null? lst)
      '()
      (cons (proc (car lst))
            (map proc (cdr lst)))))

(define (stream-map proc s)
  (if (stream-null? s)
      the-empty-stream
      (cons-stream (proc (stream-car s))
                   (stream-map proc (stream-cdr s)))))
```

```
(define (filter pred lst)
  (cond ((null? lst) '())
        ((pred (car lst))
         (cons (car lst)
               (filter pred (cdr lst))))
        (else (filter pred (cdr lst)))))

(define (stream-filter pred s)
  (cond ((stream-null? s) the-empty-stream)
        ((pred (stream-car s))
         (cons-stream (stream-car s)
                      (stream-filter pred (stream-cdr s))))
        (else (stream-filter pred (stream-cdr s)))))


(define (enum-interval low high)
  (if (> low high)
      '()
      (cons low (enum-interval (+ low 1) high))))

(define (enumerate-interval low high)
  (if (> low high)
      the-empty-stream
      (cons-stream low
                   (enumerate-interval (+ low 1) high))))
```

Some Simple Streams

```
;; Recursive definition
(define (integers-from n)
  (cons-stream n
               (integers-from (+ 1 n))))

(define integers (integers-from 1))


;; Examples using stream HOP
(define no-sevens
  (stream-filter (lambda (n) (not (divisible? n 7)))
                 integers))

(define (divisible? x y) (= (remainder x y) 0))

(stream-ref no-sevens 100)
;Value: 117
```

Sieve of Eratosthenes

```
;; Sieve
(define (sieve s)
  (cons-stream
    (stream-car s)
    (sieve (stream-filter
              (lambda (x)
                (not (divisible? x (stream-car s))))
              (stream-cdr s)))))


(define primes (sieve (integers-from 2)))

(stream-ref primes 200)
;Value: 1229
```

More Stream Utilities

```
(define (add-streams s1 s2)
  (cond ((stream-null? s1) s2)
        ((stream-null? s2) s1)
        (else
         (cons-stream (+ (stream-car s1) (stream-car s2))
                      (add-streams (stream-cdr s1)
                                   (stream-cdr s2))))))

(define (scale-stream c s)
  (stream-map (lambda (x) (* x c)) s))

(define (show-stream s n)
  (cond ((= n 0) 'done)
        (else (write-line (stream-car s))
              (show-stream (stream-cdr s) (- n 1)))))

(define (stream-map2 proc s1 s2)
  (if (stream-null? s1)
      the-empty-stream
      (cons-stream (proc (stream-car s1)
                         (stream-car s2))
                   (stream-map2 proc
                                (stream-cdr s1)
                                (stream-cdr s2)))))
```

Stream of Fibonacci numbers

```
(define fibs
  (cons-stream
    0
    (cons-stream
      1
      (add-streams (stream-cdr fibs)
                   fibs))))

(show-stream fibs 9)
0
1
1
2
3
5
8
13
21
```

Square Roots

```
;; Previous procedural implementation
(define (sqrt x)
  (define (try guess)
    (if (good-enough? guess)
        guess
        (try (improve guess))))
  (define (improve guess)
    (average guess (/ x guess)))
  (define (good-enough? guess)
    (close? (* guess guess) x))
  (try 1))
```

Stream of Square Roots

```
(define (sqrt-improve guess x)
  (average guess (/ x guess)))
(define (average a b) (/ (+ a b) 2))

(define (sqrt-stream x)
  (cons-stream 1.0
               (stream-map (lambda (g)
                             (sqrt-improve g x))
                           (sqrt-stream x))))

(show-stream (sqrt-stream 2) 7)
1.
1.5
1.4166666666666665
1.4142156862745097
1.4142135623746899
1.414213562373095
1.414213562373095
;Value: done
```

Stream to Desired Tolerance Limit

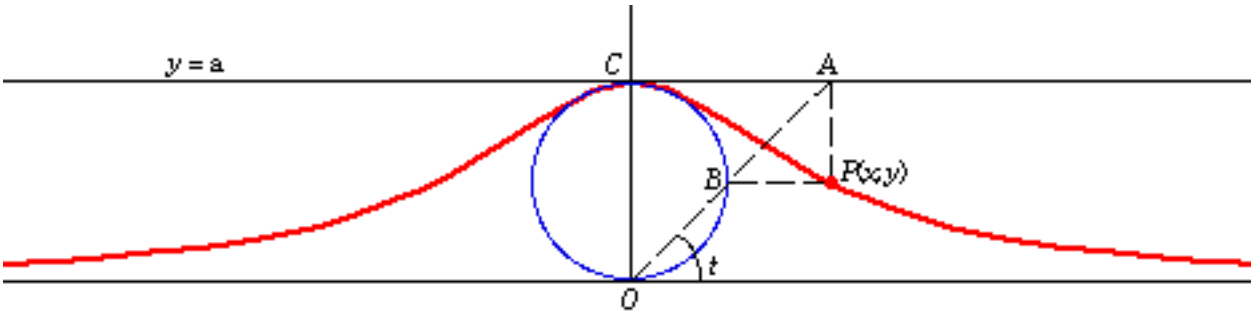
```
(define (stream-limit s tol)
  (define (iter s)
    (let ((f1 (stream-car s))
          (f2 (stream-car (stream-cdr s))))
      (if (close-enuf? f1 f2 tol)
          f2
          (iter (stream-cdr s)))))
  (iter s))

(define (close-enuf? x y tol)
  (< (abs (- x y)) tol))

(stream-limit (sqrt-stream 2) 1.e-5)
;Value: 1.4142135623746899
```

Witch of Agnesi

```
(define (witch x)
  (/ 4 (+ 1 (* x x))))
```



The bell-shaped witch of Maria Agnesi can be constructed in the following way. Start with a circle of diameter a, centered at the point (0,a/2) on the y-axis. Choose a point A on the line y = a and connect it to the origin with a line segment. Call the point where the segment crosses the circle B. Let P be the point where the vertical line through A crosses the horizontal line through B. The witch is the curve traced by P as A moves along the line y = a.

Trapezoidal Integration

```
(define (trapezoid f a b h)
  (let ((dx (* (- b a) h))
        (n (/ 1 h)))
    (define (iter i sum)
      (let ((x (+ a (* i dx))))
        (if (>= i n)
            sum
            (iter (+ i 1)
                  (+ sum (f x))))))
    (* dx
       (iter 1
             (+ (/ (f a) 2)
                (/ (f b) 2))))))

(define (witch x)
  (/ 4 (+ 1 (* x x))))

(trapezoid witch 0. 1. .1)
;Value: 3.1399259889071587

(trapezoid witch 0. 1. .01)
;Value: 3.141575986923129
```

Stream of Approximations to π

```
(define (keep-halving R h)
  (cons-stream (R h)
               (keep-halving R (/ h 2))))

(show-stream (keep-halving
              (lambda (h) (trapezoid witch 0 1 h))
              0.1)
             10)
3.1399259889071587
3.1411759869541287
3.1414884869236115
3.141566611923134
3.1415861431731273
3.1415910259856252
3.1415922466887523
3.1415925518645325
3.1415926281584774
3.1415926472319597
;Value: done

(stream-limit
 (keep-halving (lambda (h) (trapezoid witch 0 1 h))
               0.5)
 1.e-9)
;Value: 3.1415926534345684

;; This requires 65,549 evaluations of the witch
```

Accelerating the Stream

Given a sequence of values:

$$R(0),\ R(h),\ R(h/2),\ R(h/4),\ \ldots$$

If we know that R has the form

$$R(h) = A + Bh^p + Ch^{2p} + Dh^{3p} + \ldots$$

Then:

$$\frac{2^p R\left(\frac{h}{2}\right) - R(h)}{2^p - 1} = A + C_x h^{2p} + D_2 h^{3p} + \ldots$$

```
(define (accel-halving-seq s p)
  (let ((2^p (expt 2 p)))
    (let ((2^p-1 (- 2^p 1)))
      (stream-map2 (lambda (Rh Rh/2)
                     (/ (- (* 2^p Rh/2)
                           Rh)
                        2^p-1))
                   s
                   (stream-cdr s)))))
```

Rapid Acceleration

```
(define (make-tableau s p)
  (define (rows s order)
    (cons-stream s
                 (rows (accel-halving-seq s order)
                       (+ order p))))
  (rows s p))

(define (richardson-accel s p)
  (stream-map stream-car (make-tableau s p)))

(show-stream
  (richardson-accel
    (keep-halving (lambda (h) (trapezoid witch 0 1 h)) .1)
    2)
  5)
3.1399259889071587
3.1415926529697855
3.1415926536207945
3.141592653589793
3.1415926535897944

(stream-limit (richardson-accel
               (keep-halving
                (lambda (h) (trapezoid witch 0 1 h))
                .1)
               2)
              1.e-9)
;Value: 3.1415926536207945
```

This requires only 73 evaluations of the witch!