Announcements
Efficient Sequence Processing
Sequence Operations

Map, filter, and reduce express sequence manipulation using compact expressions

Example: Sum all primes in an interval from \(a\) (inclusive) to \(b\) (exclusive)

```python
def sum_primes(a, b):
    total = 0
    x = a
    while x < b:
        if is_prime(x):
            total = total + x
        x = x + 1
    return total
```

```python
def sum_primes(a, b):
    return sum(filter(is_prime, range(a, b)))
```

Demo

Space: Constant

Also Constant
Streams
Streams are Lazy Scheme Lists

A stream is a list, but the rest of the list is computed only when needed:

```
(car (cons 1 nil)) -> 1  (car (cons-stream 1 nil)) -> 1
(cdr (cons 1 nil)) -> ()  (cdr-stream (cons-stream 1 nil)) -> ()
(cons 1 (cons 2 nil))  (cons-stream 1 (cons-stream 2 nil))
```

Errors only occur when expressions are evaluated:

```
(cons 1 (cons (/ 1 0) nil))  -> ERROR
(cons-stream 1 (cons-stream (/ 1 0) nil))  -> (1 . #[promise (not forced)])
(car (cons-stream 1 (cons-stream (/ 1 0) nil))) -> 1
(cdr-stream (cons-stream 1 (cons-stream (/ 1 0) nil))) -> ERROR
(Demo)
```
Stream Ranges are Implicit

A stream can give on-demand access to each element in order

```
(define (range-stream a b)
  (if (>= a b)
      nil
      (cons-stream a (range-stream (+ a 1) b))))

(define lots (range-stream 1 10000000000000000000))

scm> (car lots)
1
scm> (car (cdr-stream lots))
2
scm> (car (cdr-stream (cdr-stream lots)))
3
```
Infinite Streams
Integer Stream

An integer stream is a stream of consecutive integers.

The rest of the stream is not yet computed when the stream is created.

\[
(\text{define} \ (\text{int-stream} \ \text{start}) \n\quad (\text{cons-stream} \ \text{start} \ (\text{int-stream} \ (+ \ \text{start} \ 1))))
\]
Stream Processing

(Demo)
Recursively Defined Streams

The rest of a constant stream is the constant stream

\[
(\text{define} \ \text{ones} \ (\text{cons-stream} \ 1 \ \text{ones}))
\]

Combine two streams by separating each into car and cdr

\[
(\text{define} \ (\text{add-streams} \ s \ t) \\
\quad (\text{cons-stream} \ (+ \ (\text{car} \ s) \ (\text{car} \ t)) \\
\quad \quad (\text{add-streams} \ (\text{cdr-stream} \ s) \\
\quad \quad \quad (\text{cdr-stream} \ t))))
\]

\[
(\text{define} \ \text{ints} \ (\text{cons-stream} \ 1 \ (\text{add-streams} \ \text{ones} \ \text{ints})))
\]
Higher-Order Stream Functions
Higher-Order Functions on Streams

Implementations are identical, but change cons to cons-stream and change cdr to cdr-stream

```
(define (map-stream f s)
  (if (null? s)
      nil
      (cons-stream (f (car s))
                   (map-stream f (cdr-stream s))))
)

(define (filter-stream f s)
  (if (null? s)
      nil
      (if (f (car s))
          (cons-stream (car s)
                       (filter-stream f (cdr-stream s)))
          (filter-stream f (cdr-stream s))))
)

(define (reduce-stream f start)
  (if (null? s)
      start
      (reduce-stream f (cdr-stream s)
                      (f start (car s))))
)```
A Stream of Primes

For any prime \( k \), any larger prime must not be divisible by \( k \).

The stream of integers not divisible by any \( k \leq n \) is:

The stream of integers not divisible by any \( k < n \)

Filtered to remove any element divisible by \( n \)

This recurrence is called the Sieve of Eratosthenes

\[
2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
\]

(Demo)