Streams
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)))
```

```
def sum_primes(a, b):
    return sum((x for x in range(a, b) if is_prime(x)))
```

Space: Constant Also Constant

(Demo)
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)) \rightarrow 1
(cdr (cons 1 nil)) \rightarrow ()
(cons 1 (cons 2 nil))

(car (cons-stream 1 nil)) \rightarrow 1
(cdr-stream (cons-stream 1 nil)) \rightarrow ()
(cons-stream 1 (cons-stream 2 nil))

Errors only occur when expressions are evaluated:

(cons 1 (cons (/ 1 0) nil)) \rightarrow ERROR
(cons-stream 1 (cons-stream (/ 1 0) nil)) \rightarrow (1 . #[promise (not forced)])
(car (cons-stream 1 (cons-stream (/ 1 0) nil))) \rightarrow 1
(cdr-stream (cons-stream 1 (cons-stream (/ 1 0) nil))) \rightarrow 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

```
(define (int-stream start)
  (cons-stream start (int-stream (+ 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

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

\[(\text{define} \ \text{ints} \ (\text{cons-stream} \ 1 \ (\text{add-streams} \ \text{ones} \ \text{ints})))\]
Example: Repeats

```scheme
(define a (cons-stream 1 (cons-stream 2 (cons-stream 3 a))))

(define (f s) (cons-stream (car s)
    (cons-stream (car s)
    (f (cdr-stream s))))))

(define (g s) (cons-stream (car s)
    (f (g (cdr-stream s))))))

What's (prefix a 8)?
  ( __  __  __  __  __  __  __  __ )

What's (prefix (f a) 8)?
  ( __  __  __  __  __  __  __  __ )

What's (prefix (g a) 8)?
  ( __  __  __  __  __  __  __  __ )
```
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

\[
\begin{align*}
2, &\ 3, &\ 4, &\ 5, &\ 6, &\ 7, &\ 8, &\ 9, &\ 10, &\ 11, &\ 12, &\ 13
\end{align*}
\]

(Demo)