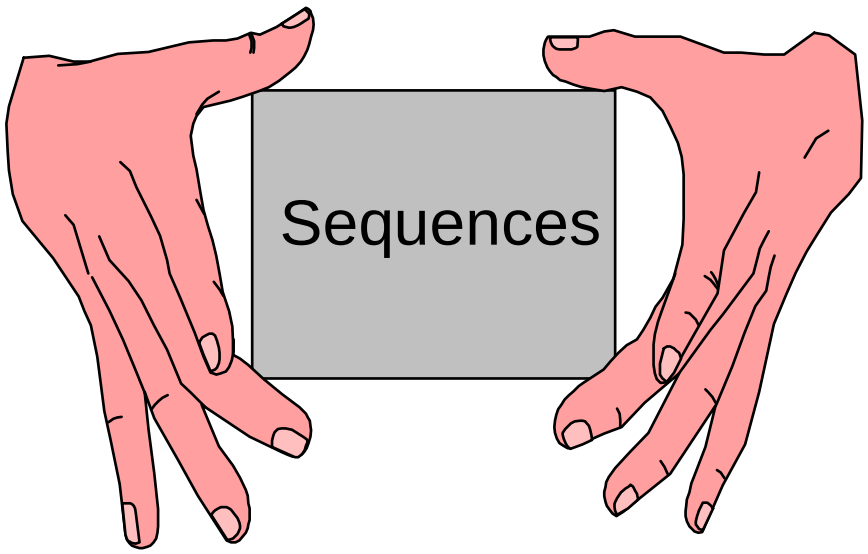
Standard ML



ML Sequences.1

Sequences, Lazy Lists

- Characteristics of lazy lists
  - Elements are not evaluated until their values are required
  - May be infinite
- Expressing lazy lists in ML

```
datatype 'a seq = Nil
                | Cons of 'a * (unit-> 'a seq)
fun head(Cons(x,_)) = x;
> val head = fn : 'a seq -> 'a
fun tail(Cons(_,xf)) = xf();
> val tail = fn : 'a seq -> 'a seq
```

- ML evaluates the *E* expression in `Cons(x,`*E*`)`, so to obtain lazy evaluation we must write `Cons(x,fn()=>`*E*`)`.

ML Sequences.2

Examples Of Sequences

- An increasing sequence of integers

```
fun from k = Cons(k,fn()=>from(k+1));
> val from = fn : int -> int seq
from 1;
> val it = Cons (1, fn) : int seq
tail it;
> val it = Cons (2, fn) : int seq
```

- A sequence of squared integers

```
fun squares Nil : int seq = Nil
    | squares (Cons(x,xf)) =
              Cons(x*x, fn()=> squares (xf()));
> val squares = fn : int seq -> int seq
squares (from 1);
> val it = Cons (1, fn) : int seq
head(tail(tail(tail(tail it))))
> val it = 25 : int
```

ML Sequences.3

Elementary Sequence Processing

- Adding two sequences
  - ```
    fun addq (Cons(x,xf), Cons(y,yf))=
          Cons(x+y, fn()=>addq(xf(),yf()))
      | addq _ : int seq  =  Nil;
    ```

    > *val addq = fn: int seq * int seq -> int seq*

- Appending two sequences
  - ```
    fun appendq (Nil,        yq) = yq
      | appendq (Cons(x,xf), yq) =
          Cons(x,fn()=>appendq(xf(),yq));
    ```

    > *val appendq = fn : 'a seq * 'a seq -> 'a seq*

  - appendq (xq, yq) :
    No elements of yq appear in the output unless xq is finite !

ML Sequences.4

Functionals for Sequences

- Interleaving two sequences
  - ```
    fun interleaving (Nil, yq)       = yq
      | interleaving (Cons(x,xf),yq) =
          Cons(x, fn()=>interleaving(yq,xf()));
    ```
  - > *val interleaving = fn: 'a seq * 'a seq -> 'a seq*

- `mapq` and `filterq`
  - ```
    fun mapq f Nil          = Nil
      | mapq f(Cons(x,xf)) = Cons(f x,fn()=>mapq f(xf()));
    ```
    - > *val mapq = fn : ('a ->'b) -> 'a seq -> 'b seq*
  - ```
    fun filterq pred Nil          = Nil
      | filterq pred (Cons(x,xf)) =
          if pred x then Cons(x,fn()=>filterq pred(xf()))
                    else filterq pred (xf());
    ```
    - > *val filterq = fn : ('a ->bool) -> 'a seq -> 'a seq*

ML Sequences.5

Example: Prime numbers

- ```
  fun notDivide p = filterq (fn n => n mod p <> 0);
  ```
  - > *val notDivide = fn : int -> int seq -> int seq*
  - notDivide p builds a filter that removes all numbers that are divisible by p
- ```
  fun sieve (Cons(p,nf)) =
  Cons(p, fn () => sieve(notDivide p (nf())));
  ```
  - > *val sieve = fn : int seq -> int seq*
  - Sieve receives a sequence and returns a sequence in which every head places a notDivide filter on the tail; therefore the tail never has numbers that are divisible by any number that has ever been in the head.
- ```
  val primes = sieve (from 2);
  ```
  - > *val primes = Cons (2, fn) : int seq*

ML Sequences.6