(define for-all? 
  (lambda (lst pred?) 
    ...?...
  )

> (for-all? '(5 2 9 4) positive?) #t
> (for-all? '(5 2 -9 4) positive?) #f
> (for-all? '(5 2 9 4) number?) #t
> (for-all? '(5 b 9 4) number?) #f
> (for-all? '() number?) #t

for-all? is a procedure that takes a list lst and a predicate pred? as inputs.

The predicate pred? should accept one argument.

for-all? returns #t if there is no member m of lst for which (pred? m) would return #f.

Otherwise, for-all? returns #f.

Universal quantification: (\( \forall \ x \) \( P(x) \)).
(define for-all?  
(lambda (lst pred?)  
  (not (member #f (map pred? lst))))))

This gives the same answer as the previous definition, but it will stop as soon as pred? is false for an element of lst.

There's a built-in function that has early termination like our recursive for-all?, called andmap.

Note: Inputs are switched:

> (andmap number? '(1 2 3))
#t

Example: all-above?
(define all-above? (lambda (lst n) ...?...))

> (all-above? '(5 2 9 4) 0)
#t
> (all-above? '(5 2 -9 4) -10)
#t
> (all-above? '(5 2 9 4) 5)
#f
> (all-above? '() 5)
#t

(define all-above?
  (lambda (lst n)
    (for-all? lst (lambda (x) (> x n)))))

Functional pattern:
True for some element?

(define there-exists?
  (lambda (lst pred?) ...?...))

> (there-exists? '(-5 -2 9 -4) positive?)
#t
> (there-exists? '(-5 -2 -9 -4) positive?)
#f
> (there-exists? '(a z 0 1) number?)
#t
> (there-exists? '(a b c d) number?)
#f
> (there-exists? '() number?)
#f
The **there-exists?** procedure

**there-exists?** takes a list `lst` and a predicate `pred?` as inputs.

The predicate `pred?` should accept one argument.

**there-exists?** returns #t if there is at least one member `m` of `lst` such that `(pred? m)` would return #t.

Otherwise, **there-exists?** returns #f.

*Existential quantification:* $(\exists x) \, P(x)$.

```
(define there-exists? (lambda (lst pred?)
    (member? #t (map pred? lst))))
```

```
(define there-exists? (lambda (lst pred?)
    (and (not (null? lst))
        (or (pred? (first lst))
            (there-exists? (rest lst)
                pred?))))))
```

Example: **some-below?**

This gives the same answer as the previous definition, but it behaves differently. How?
(define some-below?
  (lambda (lst n)
    ...
    )))

> (some-below? '(5 2 9 4) 4)
#t
> (some-below? '(5 2 -9 4) -10)
#f
> (some-below? '(5 2 -9 4) -3)
#t
> (some-below? '(5 2 9 4) 5)
#t
> (some-below? '() 5)
#f

(define some-below?
  (lambda (lst n)
    (there-exists? lst
      (lambda (x) (< x n))))))

Example: common-factor?

Does a list of integers contain a single common factor of all its members?

(define common-factor?
  (lambda (lst)
    ...
    )))

> (common-factor? '(18 6 42 72))
#t
> (common-factor? '(25 36 8 49))
#f
Sub-problem: Does one integer divide another integer evenly?

(define divides?
  (lambda (dividend divisor)
    (= 0 (remainder dividend divisor))))

(define common-factor?
  (lambda (lst)
    (there-exists? lst
      (lambda (divisor)
        (for-all? lst
          (lambda (dividend)
            (divides? divisor dividend)))))))

Acknowledgments

This lecture incorporates material from:

  Tom Ellman
  Luke Hunsberger