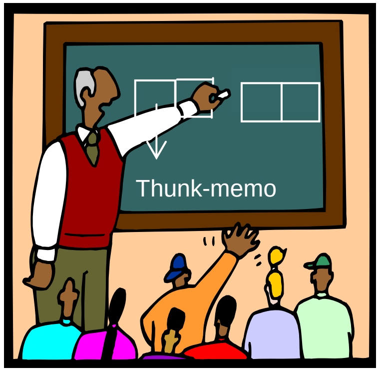
6.001 recitation 21 5/04/07

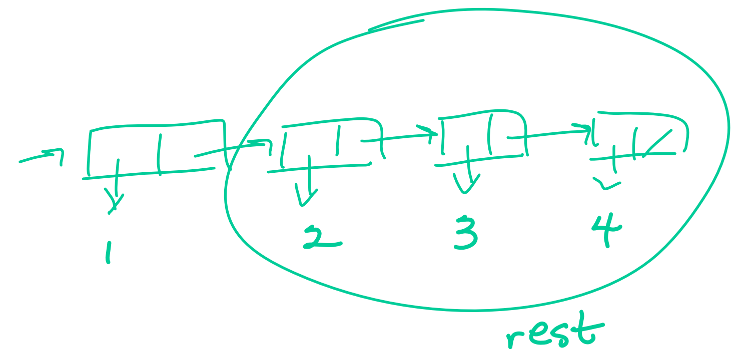
- streams

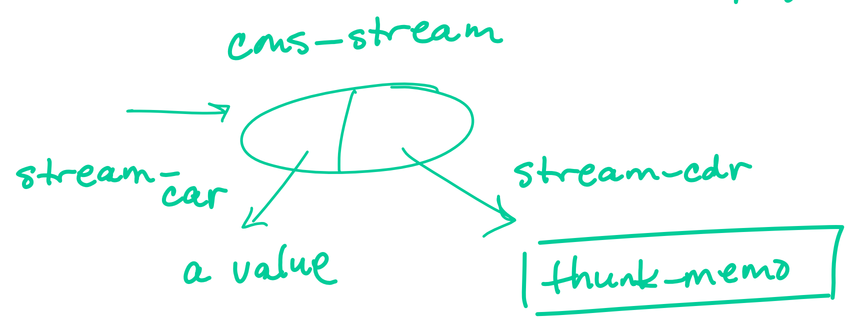


Dr. Kimberle Koile

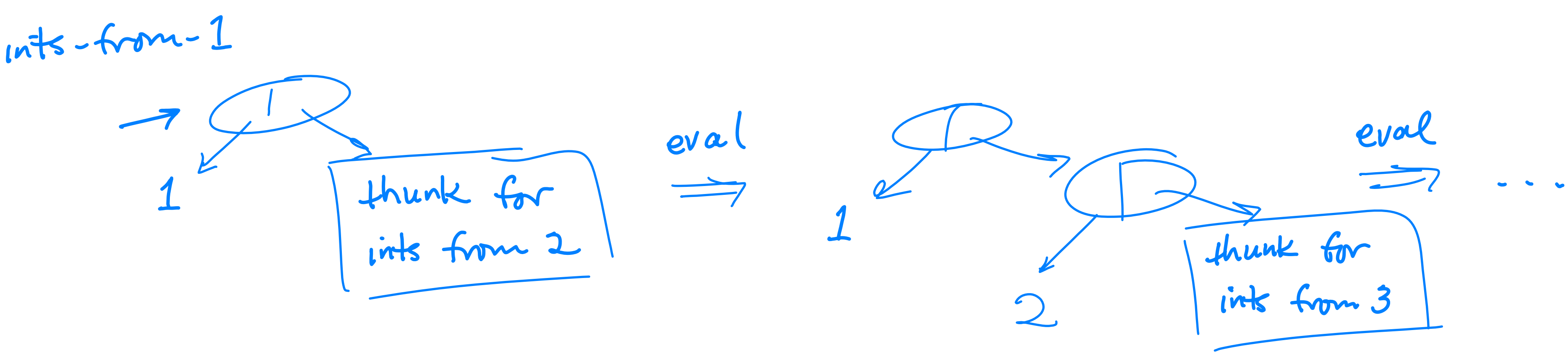
delayed lists: streams

```
(define (cons-my-list first rest)
   (cons first rest))

(define ints-from-1-to-4   (cons-my-list 1 '(2 3 4)))

(define ints-from-1    ???)
```



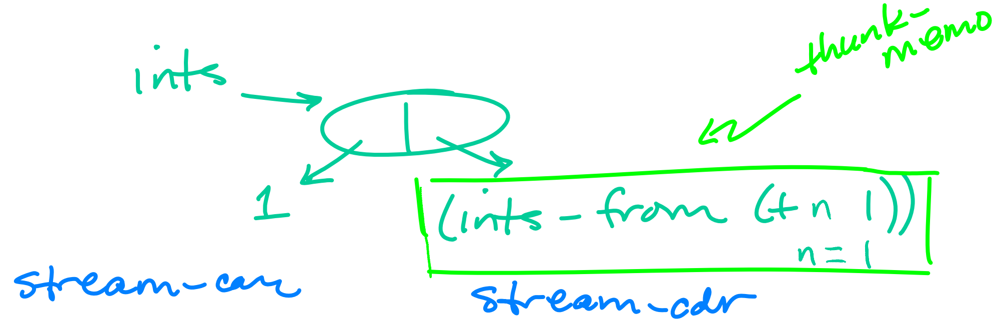


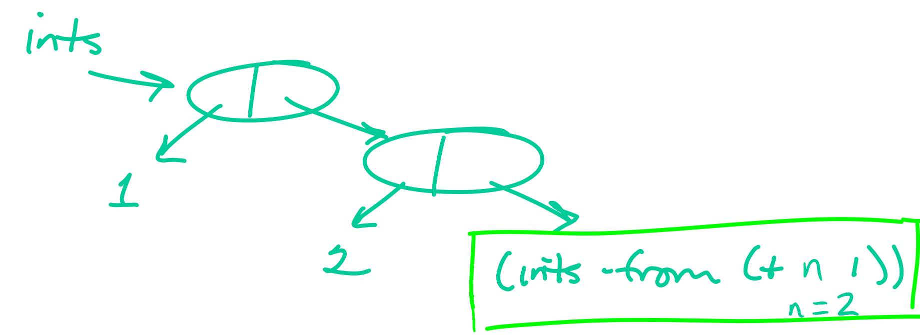

```
(define (cons-stream first (rest (lazy-memo))
      (cons first rest))
```



delayed lists: streams

```
(define (ints-from-n n)
   (cons-stream n (ints-from (+ n 1))))

(define ints  (ints-from 1))
```





(stream-car ints) ⇒ 1

(stream-cdr ints) ⇒ force-eval of:

- (ints-from (+ n 1)) ; n=1
- (ints-from 2)
- (cons-stream 2 [(ints-from (+ n 1)) n=2] ← thunk-memo

streams summary

Key ideas:

- streams are *delayed lists*
- represent a stream as a *cons-strea*m, pair-like object with lazy cdr
- define a stream by figuring out *first* element, then how to compute *rest*

Examples:

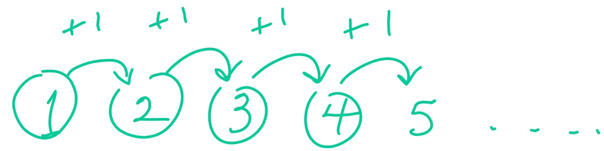
- integers 1, 2, 3, 4, 5 ….
- factorials 1, 2, 6, 24, 120 ...

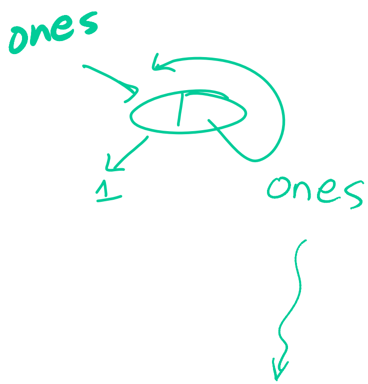
another way to think about streams

ints

first element? 1

rest?



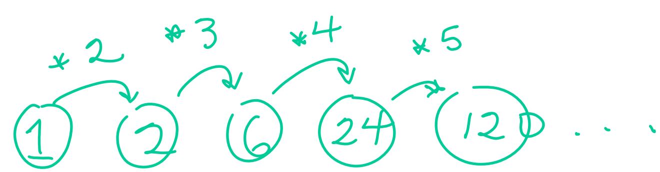



```
(define ints
    (cons-stream 1 (add-streams ints ones)))
```

facts

first element? 1

rest?




```
(define facts
    (cons-stream 1 (mult-streams facts
                        (stream-cdr ints))))
```

summary of examples of defining streams

finite

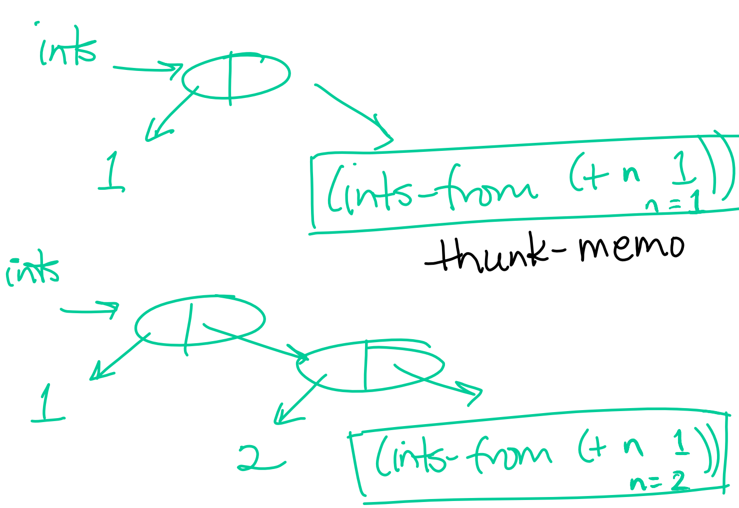
(stream-interval 1 1×10^{100})

```
(define (stream-interval a b)
  (cons-stream a (stream-interval (+ a 1) b)))
```

infinite (indefinite)

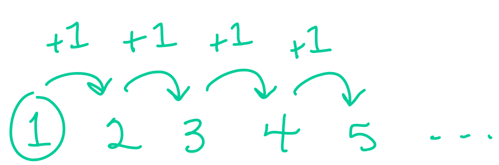
explicit

```
(define (ints-from n)
  (cons-stream n (ints-from (+ n 1))))

(define ints (ints-from 1))
```



implicit

```
(define ints (cons-stream
               1
               (add-streams ints ones))
```



useful stream procedures

- stream-filter
- stream-map
  - 2 args
  - variable number args
- stream-ref
- add-streams
- mult-streams

```
(define (any? test l)
  (cond ((null? l) #f)
        ((test (car l)) #t)
        (else (any? test (cdr l)))))

(define (map proc . args)
  (if (null? args)
      nil
      (if (any? null? args)
          nil
          (cons (apply proc (map car args))
                (apply map (cons proc (cdr args)))))))

(define (stream-map proc . args)
  (if (null? args)
      nil
      (if (any? null? args)
          nil
          (cons-stream (apply proc (map stream-car args))
                       (apply stream-map (cons proc (map stream-cdr args)))))))
```

Does this procedure work?

yes

```
(define (add-streams s1 s2)
  (cons-stream (+ (stream-car s1) (stream-car s2))
               (add-streams (stream-cdr s1) (stream-cdr s2))))
```

first elt

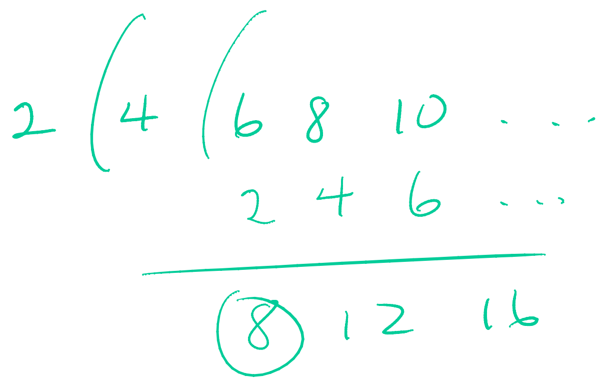
rest

another example

What value is printed in response to the last expression in this sequence of expressions?

```
(define evens  (cons-stream 2 (stream-map  (lambda (x) (+ x 2))  evens))

(stream-car
   (add-streams  evens   (stream-cdr  (stream-cdr  evens))))))
```

8



Problem 1

1. **Write mult-stream which takes two streams and returns a new stream that is the product of the two streams.**

```
(define (mult-streams s1 s2)

    (cons-stream (* (stream-car s1) (stream-car s2))
                 (mult-streams (stream-cdr s1) (stream-cdr s2)))

)
```

Problem 2

2. Write stream-ref, modeled after list-ref, which takes a stream and a number n and returns the nth element of the stream.

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

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

Problem 3 (from a previous final exam)

3. Write list->stream, which turns a list into a stream.

```
(define (list->stream l)

        (cons-stream (car l) (list->stream (cdr)))

)
```

Problem 4 (modified from a previous final exam problem)

4. Assume that the following have been evaluated:

```
(define ones  (cons-stream  1  ones))

(define (add-streams s1  s2)
  (cons-stream  (+ (stream-car  s1)  (stream-car  s2))
                (add-streams  (stream-cdr  s1) (stream-cdr  s2))))
```

Consider the expression:

```
(define integers (add-streams  ones  integers))
```

For each of the following, put an X in the box if the statement applies to the above scenario:

- [ ] The expression evaluates to a stream of integers.
- [ ] The interpreter goes into an infinite loop when (stream-cdr integers) is evaluated.
- [ ] An "unbound variable" error occurs when the above expression defining ones is evaluated.
- [x] An "unbound variable" error occurs when the above expression defining integers is evaluated.

because the second arg to add-streams is not delayed

Problem 5 (from a previous final exam)

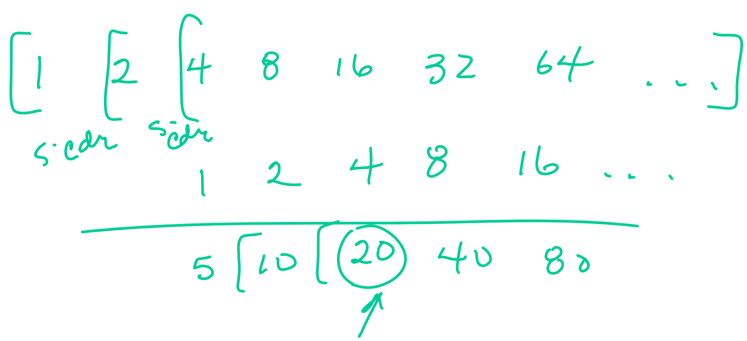
5. What value is printed in response to the last expression in this sequence of expressions?

powers of 2

```
(define s   (cons-stream 1  (stream-map  (lambda (x) (* x 2))  s)))
```

```
(stream-car
   (stream-cdr
      (stream-cdr
         (add-streams  s  (stream-cdr  (stream-cdr  s))))))
```

20



Problem 6

Consider the sequence of expressions:

```
(define (stream-enumerate-interval low high)
  (if (> low high)
      the-empty-stream
      (cons-stream  low  (stream-enumerate-interval  (+  low  1)  high))))
(define  sum  0)
(define (accum x)
  (set! sum (+ x sum))
  sum)
(define seq  (stream-map accum  (stream-enumerate-interval  1  10)))
(define y  (stream-filter even? seq))
(define z  (stream-filter  (lambda (x)  (=  (remainder x  5)   0))  seq))
```

What is the printed response to evaluating the following expressions. Assume print-stream prints out stream elements inside [ ], e.g. [1 2 3]

[1 2 3 4 5 6 7 8 9 10]
sum 0 [1 3 6 10 15 21 28 36 45 55]

6a. (print-stream y)

[6 10 28 36]

6b. (stream-ref y 3)

36

6c. (print-stream z)

[10 15 45 55]

Problem 7

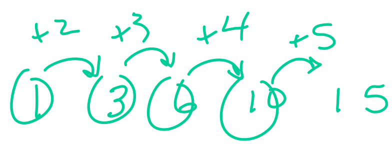
Assume that we're interested in the partial sums of a stream. Given a stream S, for example, a stream of partial sums for S is the stream S0, S0 + S1, S0 + S1 + S2, ….

7a. Write an expression that defines a stream that is the partial sum of integers
For example, (partial-sums integers) should be the stream 1, 3, 6, 10, 15 …

(define ints (cons-stream 1 (add-streams ints ones)))

(define int-partial-sums

```
(cons-stream 1
      (add-streams int-partial-sums
                   (stream-cdr ints)))
```



)

7b. Write a procedure partial-sums that takes a stream as an argument, and returns the stream S0, S0 + S1, S0 + S1 + S2 …. For example, (partial-sums integers) should be the same stream as in part a.

(define (partial-sums s)

```
(cons-stream (stream-car s)
             (add-streams (partial-sums s)
                          (stream-cdr s)))
```

)

Problem 8 (from a previous final exam)

8. Suppose you are given two streams and you need to produce a stream that contains both. Translating append, which works on lists, into an append-stream procedure by changing the data abstraction selectorsand constructor will not work if the streams are indefinite in length: "appending" the infinite streamS1, S2, S3 … and a second infinite stream T1, T2, T3 … results in the stream S1 S2, S3, …, T1, T2, T3, … which is effectively the same a the first stream. The solution is to merge the two streams instead of appending them. Write a procedure called alternate-streams that consumes two streams and returns a single one that contains elements alternating from the two inputs.

Remember that the data abstraction for streams uses stream-null?, stream-car, stream-cdr, null-stream, and cons-stream.

```
(define (alternate streams s1 s2)
    (if (stream-null? s1)
        s2
        (cons-stream (stream-car s1)
                     (alternate-streams s2 (stream-cdr s2)))))

or

 (define (alternate-streams s1 s2)
    (if (stream-null? s1)
        s2
        (cons-stream (stream-car s1)
                     (cons-stream (stream-car s2)
                                  (alternate-streams (stream-cdr s1)
                                                     (stream-cdr s2))))))
)
```