Lab 11 solutions

LAB
===

1. The result of (delay ...) is of type *promise*.
The result of (force (delay ...)) is of whatever type the "..." would be without delay; in this case it's (force (delay (+ 1 27))) and that's of type integer.

2. (stream-cdr (stream-cdr (cons-stream 1 '(2 3)))) causes an error because (2 3) isn't a stream.

```
   (stream-cdr (cons-stream 1 '(2 3)))        -->   (2 3)
        (stream-cdr '(2 3))                   -->   (force (cdr '(2 3)))
                                          -->   (force '(3))
```

but (3) isn't a promise!

3. (delay (enumerate-interval 1 3)) returns a promise.
(stream-enumerate-interval 1 3) returns a stream, i.e., a pair whose car is 1 and whose cdr is a promise.

The only thing you can do with the result of (delay ...) is FORCE it; you can't ask for its stream-car or stream-cdr, etc. By contrast, with the stream, you can stream-car or stream-cdr it, but *not* force it, because it's not a promise.

4. the 4-2-1 stream.

```
(define (next-num n)
  (if (even? n)
        (/ n 2)
        (+ (* n 3) 1)))

(define (num-seq n)
  (cons-stream n (num-seq (next-num n))))

(define (seq-length s)
  (if (= (stream-car s) 1)
        1
        (+ 1 (seq-length (stream-cdr s)))))
```

CS 61A Week 13 Lab Solutions

1. We make two changes: First, change the mapper function so that the intermediate pairs use the play names as keys. This will result in a separate reduce instance for each play. Then, we do an ordinary non-parallel stream-accumulate to combine those results into a single count. (Shakespeare didn't write so many plays that we can't accumulate the results of the different plays on a single processor.)

```
(STREAM-ACCUMULATE (LAMBDA (NEW OLD)
                   (+ (KV-VALUE NEW) OLD))
                   0
                   (mapreduce (lambda (kv-pair)
                                (list (make-kv-pair (KV-KEY KV-PAIR) 1)))
                               +
                               0
                               "/gutenberg/shakespeare"))
```

2(a). The mapper gets a line as input, but wants to output a separate key-value pair for each word in the line.

```
(define gutenberg-wordcounts
  (mapreduce (lambda (line-pair)
                   (map (lambda (wd) (make-kv-pair wd 1))
                             (kv-value line-pair)))
          +
          0
          "/gutenberg/shakespeare"))
```

2(b). We want to reduce gutenberg-wordcounts by comparing the counts of two words at a time, keeping the one with the larger count. But there are a lot of words in English, so we'd like to parallelize that reduction. My solution, not perfectly efficient but simple and better than nothing, is to split up the words based on their first letters. So we do a second mapreduce operation in which the mapper turns (the . 300) into (t . (the . 300)) (using the first letter as a new key and keeping the entire original key-value pair as the new value). Mapreduce is going to return a stream of length just over 26 (the letters of the alphabet plus special characters like apostrophes, etc.), with the most common word with each initial letter; that stream is small enough that we can reduce it to the single most common word using stream-accumulate on a single computer:

```
(stream-accumulate (lambda (new old)
                    (if (> (kv-value (kv-value new))
                          (kv-value (kv-value old)))
                     new
                     old))
                    (make-kv-pair 'f (make-kv-pair 'foo 0))
                    (mapreduce
                    (lambda (kv-pair)
                    (list (make-kv-pair (first (kv-key kv-pair)) kv-pair)))
                    (lambda (new old)
                    (if (> (kv-value new) (kv-value old)) new old))
                    (make-kv-pair 'xyzzy 0)
                    gutenberg-wordcounts))
```

Note that the return value ends up looking like (t . (the . 24890)).

2(c). This one is simpler. The mapper of our new mapreduce call will act as a filter, keeping only words whose count is 1.

```
(define singletons
  (stream-map kv-key
        (mapreduce (lambda (kv-pair)
                          (if (= (kv-value kv-pair) 1)
                          (list kv-pair)
                          '()))
                     cons
                     '()
                     gutenberg-wordcounts)))
```

You may notice that a lot of these are words with unusual punctuation. Can you think of a way to remove these, too?

3(a). As in 2(c), the reduction is trivial, and we're using the mapper as a filterer.

```
(define (find-matches pattern input)
  (mapreduce (lambda (kv-pair)
        (if (match? pattern (kv-value kv-pair))
              (list kv-pair)
              '()))
        cons
        '()
        input))
```