Application-to-all

This pattern fits recursions that take a sentence as argument and return the result of doing something to every element in the sentence.

An example of the pattern is a procedure “square-all” that squares every element in the sentence.

```
(define (square-all sent)
     (if (empty? sent)
          ‘( )
          (se         (square (first sent))
               (square-all (bf sent)))))
```

The general pattern is:

```
(define (proc-applied-to-all sent)
  (if (empty? sent)
      '()
      (se (proc (first sent))
          (proc-applied-to-all (bf sent)))))
```

A similar pattern applies to words instead of sentences. This pattern has several names. Someone with a mathematical background would call it "mapping". It may also be viewed as translation, as for instance in the `digit-values` procedure from the "Roman Numerals" case study.

What are two examples of this pattern? Write code if you have time.

Example 1:

Example 2:

Filtering

What you want to do is discard all the uninteresting words in a sentence or characters in a word. Examples from lab are the `remove` and `dupls-removed` procedures.

Another example is `keep-evens` that takes a sentence of numbers and returns a sentence of the numbers that are even.

```
(define (keep-evens sent)
  (cond
    ((empty? sent) '())
    ((even? (first sent))
     (sentence (first sent) (keep-evens(butfirst sent))) )
    (else (keep-evens(butfirst sent))) ) )
```

The general pattern is:

```
(define (filtered sent)
  (cond
    ((empty? sent) '())
    ((interesting? (first sent))
     (sentence (first sent) (filtered (butfirst sent))) )
    (else (filtered (butfirst sent))) ) )
```

There is a similar pattern for words. Some people talk about "keeping" rather than "filtering". (These are probably the same people that think a glass is half full rather than half empty.)

What are two examples of this pattern? Write code if you have time.

Example 1:

Example 2: