Exercise 4.26.

Ben Bitdiddle and Alyssa P. Hacker disagree over the importance of lazy evaluation for implementing things such as `unless`. Ben points out that it's possible to implement `unless` in applicative order as a special form. Alyssa counters that, if one did that, `unless` would be merely syntax, not a procedure that could be used in conjunction with higher-order procedures. Fill in the details on both sides of the argument. Show how to implement `unless` as a derived expression (like `cond` or `let`), and give an example of a situation where it might be useful to have `unless` available as a procedure, rather than as a special form.

Answer.

To implement `unless` as a derived expression, we include syntax procedure that extract the parts of an `unless` expression, and a procedure `unless->if` that transforms `unless` expressions into `if` expression.

```
(define (unless? exp) (tagged-list? exp 'unless))

(define (condition exp) (cadr exp))

(define (usual-value exp) (caddr exp))

(define (execptional-value exp) (cadddr exp))

(define (unless->if exp)
  (make-if (condition exp)
           (execptional-value exp)
           (usual-value exp)))
```

Currently, I still have no idea of a situation where it might be useful to have `unless` available as a procedure, rather than as a special form.