;; **max-of-list** : list-of-number → number
;; Purpose: return the largest number in the argument list
(define (max-of-list a-lon) …)

With a lower bound:

```
;; bigger: number number → number
;; Purpose: return the larger of its input numbers
(define (bigger n1 n2)
  (cond
     [(<= n1 n2)     n2]
     [else           n1]
   ))

;; max-of-list: list-of-number → number
;; Purpose: return the largest number in the list
(define (max-of-list a-lon)
   (cond
     [(empty? a-lon)  0 ]
     [(cons? a-lon)
          (biggest (first a-lon) (max-of-list (rest a-lon)))]
    ))
```

;; a **nelon** (non-empty-list-of-numbers) is either
;; -- (cons f empty), where f is a number, or
;; -- (cons f r), where f is a number and r is **nelon**

The template for a program that consumes a **nelon**:

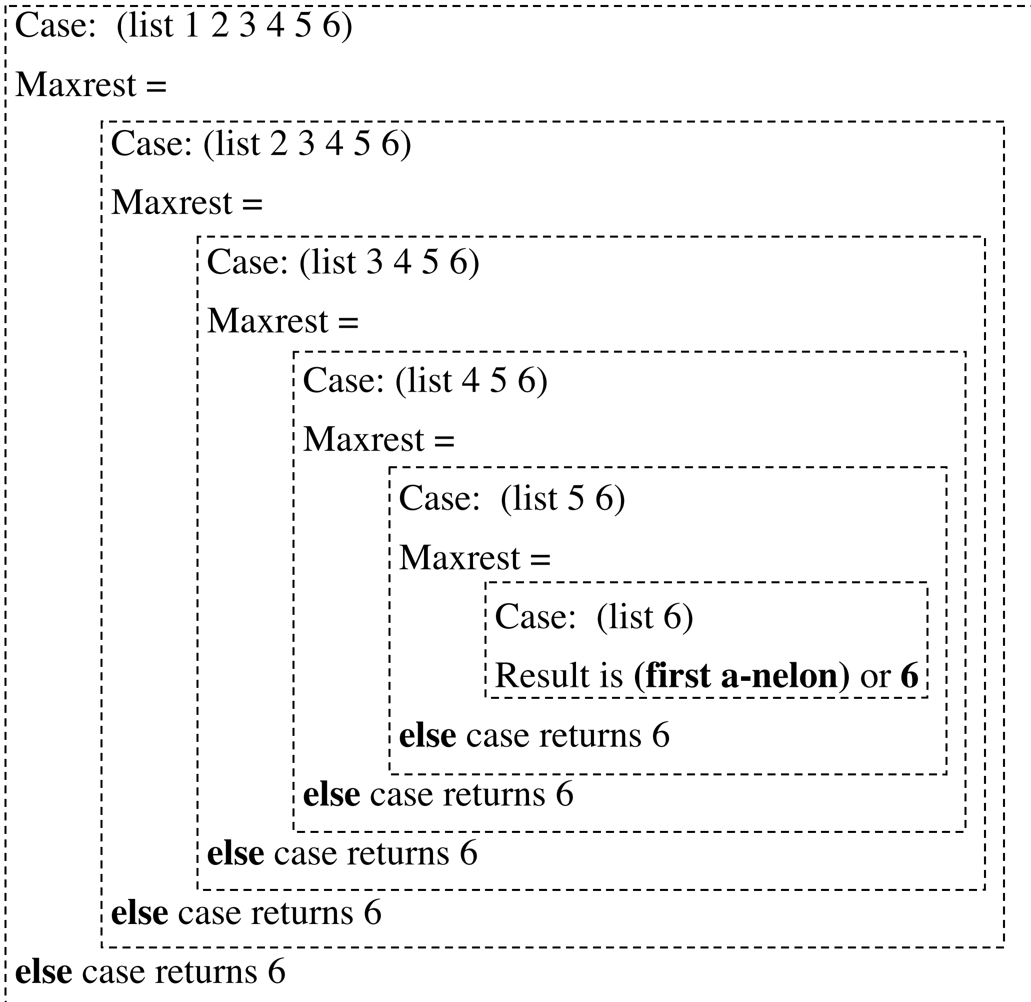
```
(define (f a-nelon)
  (cond
    [(empty? (rest a-nelon))    … (first a-nelon)]
    [(cons?   (rest a-nelon))
              … (first a-nelon) … (f (rest a-nelon))]
  ))
```

And**, max-of-list** over **nelon**s

```
;; max-of-list : nelon -> number
        ;; Purpose: returns the largest number in the input nelon
        (define (max-of-list  a-nelon)
          (cond
             [(empty? (rest a-nelon))   (first a-nelon)]
             [(cons?   (rest a-nelon))
                (cond
                     [(> (first a-nelon)  (max-of-list (rest a-nelon)))
                      (first a-nelon)]
                     [else (max-of-list (rest a-nelon))]
              )]
           ))
```

max-of-list over **nelon**s, using **local**

```
;; max-of-list : nelon -> number
;; Purpose: returns the largest number in the input nelon
(define (max-of-list  a-nelon)
  (cond
     [(empty? (rest a-nelon))   (first a-nelon)]
     [(cons?    (rest a-nelon))
       (local
          (   (define maxrest   (max-of-list (rest a-nelon))))
                (cond
                     [(> (first a-nelon)  (maxrest)) (first a-nelon)]
                   [else maxrest ] ))
        )
     ]
   ))
```



outer **cond** in the original instance of **max-of-list** returns 6 from the evaluation of the **local** in the **cons?** case