

Question 1

 [Bookmark this page](#)

Here is our nearly complete design of `filter2` from the previous two sections:

```
;; ListOfNumber -> ListOfNumber
;; produce list with only positive? elements of lon
(check-expect (positive-only empty) empty)
(check-expect (positive-only (list 1 -2 3 -4)) (list 1 3))

(define (positive-only lon) empty) ;stub

(define (positive-only lon)
  (filter2 positive? lon))

;; ListOfNumber -> ListOfNumber
;; produce list with only negative? elements of lon
(check-expect (negative-only empty) empty)
(check-expect (negative-only (list 1 -2 3 -4)) (list -2 -4))

(define (negative-only lon) empty) ;stub

(define (negative-only lon)
  (filter2 negative? lon))

;; given a list, produce only the elements of that list that satisfy the predicate p
(check-expect (filter2 positive? empty) empty)
(check-expect (filter2 negativie? (list 1 -5 5 -1)) (list 1 5))
(check-expect (filter2 positive? (list 1 -5 5 -1)) (list -5 -1))
(check-expect (filter2 false? (list false true false false true)) (list false false false))

(define (filter2 p lon)
  (cond [(empty? lon) empty]
        [else
         (if (p (first lon))
             (cons (first lon)
                   (filter2 p (rest lon)))
             (filter2 p (rest lon))))])
```

Now we need to complete the signature for `filter2`.

Question 1

1/1 point (graded)

Use the pieces below to construct the signature for `filter2`.

```
;; (X->Boolean) (listof X) -> (listof X)
```



(listof Boolean)

(listof X)

(listof Y)

(X->Y)

(Y->X)

(X->Boolean)





Answer:

Submit

 Show Answer

 Correct (1/1 point)

