Question 4-6

Bookmark this page

Question 4

1/1 point (graded)

Choose the part of the template that results from each highlighted part of the Type Comment.

```
;; ListOfNumber is one of:
;; - empty
;; (cons Number ListOfNumber
;; interp. a list of numbers

(define (fn-for-lon lon)
  (cond [(empty? lon) (...)]
        [else
          (... (first lon)
               (fn-for-lon (rest lon)))]))
```

✔

Explanation

`ListOfNumber` has two cases, so we need a `cond` expression. `empty` is the first case, and it is atomic distinct, so the answer clause is `(...)`.

Submit Show Answer

Answers are displayed within the problem

Question 5

1/1 point (graded)

```
;; ListOfNumber is one of:
;; - empty
;; (cons Number ListOfNumber
;; interp. a list of numbers

(define (fn-for-lon lon)
  (cond [(empty? lon) (...)]
        [else
          (... (first lon)
               (fn-for-lon (rest lon)))]))
```

✔

Explanation

`(cons Number ListOfNumber)` is compound, so we separate it into two parts, the first being `(first lon)`.

Submit Show Answer

Answers are displayed within the problem

Question 6

1/1 point (graded)

```
;; ListOfNumber is one of:
;; - empty
;; (cons Number ListOfNumber
;; interp. a list of numbers

(define (fn-for-lon lon)
  (cond [(empty? lon) (...)]
        [else
          (... (first lon)
               (fn-for-lon (rest lon)))]))
```

✔

Explanation

The second part of the compound is `(rest lon)` which is ListOfNumber. This is a self-reference in the type comment, so `(rest lon)` is wrapped in a call to `fn-for-lon` to form a natural recursion.

Submit Show Answer

Answers are displayed within the problem

Previous Next