Questions 1-3

Bookmark this page

Consider the following two functions:

```
;; ListOfNumber -> ListOfNumber
;; produce list with only postivie? 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)
  (cond [(empty? lon) empty]
        [else
         (if (positive? (first lon))
             (cons (first lon)
                   (positive-only (rest lon)))
             (positive-only (rest 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)
  (cond [(empty? lon) empty]
        [else
         (if (negative? (first lon))
             (cons (first lon)
                   (negative-only (rest lon)))
             (negative-only (rest lon)))]))
```

You want to design an abstract function called `filter2` based on these two functions.

Question 1

1/1 point (graded)

Which of the following are points of variance between the two functions `positive-only` and `negative-only`?:

□ the `cond` questions

□ the result of the base case

□ the structure of the else case

☑ the predicate used to decide if an element remains in the list

✓

Explanation

`positive-only` keeps just the `positive?` elements of the list, while `negative-only` keeps just the `negative?` elements of the list, so the predicates used to make this decision, `positive?` and `negative?` differ.

Submit

Show Answer

Answers are displayed within the problem

Question 2

1/1 point (graded)

What is the correct function definition for `filter2`?

○

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

○

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

○

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

●

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

✓

Explanation
We first make a copy of one of the functions with the more general name, `filter2`. Next we must add a paramater for varying position, then use that paramater in the varying position. Finally, we must replace calls to `positive-only` and `negative-only` with calls to the new abstract function, and add the varying parameter to each recursive call.

Submit

Show Answer

Answers are displayed within the problem

Question 3

1/1 point (graded)

Now that we have the abstract function `filter2`, what should be the new function body of `positive-only`?

○

```
(define (positive-only p lon)
  (filter2 p lon))
```

○

```
(define (positive-only lon)
  (filter2 p lon))
```

●

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

○

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

✓

Explanation
We call the new abstract function `filter2` with the appropriate predicate p in the function body.

Submit

Show Answer

Answers are displayed within the problem

Previous   Next

Accessibility Policy