Question 2

Bookmark this page

The following is a nearly completed design of an abstract function from two examples of repetetive code. All that's missing is the signature.

```
;; ListOfNumber -> Boolean
;; produce true if every number in lon is positive
(check-expect (all-positive? empty) true)
(check-expect (all-positive? (list 1 -2 3)) false)
(check-expect (all-positive? (list 1 2 3)) true)

(define (all-positive? lon) (andmap2 positive? lon))

;; ListOfNumber -> Boolean
;; produce true if every number in lon is negative
(check-expect (all-negative? empty) true)
(check-expect (all-negative? (list 1 -2 3)) false)
(check-expect (all-negative? (list -1 -2 -3)) true)

(define (all-negative? lon) (andmap2 negative? lon))

;;produce true if pred produces true for every element of the list
(check-expect (andmap2 positive? empty) true)
(check-expect (andmap2 positive? (list 1 -2 3)) false)
(check-expect (andmap2 positive? (list 1 2 3)) true)
(check-expect (andmap2 negative? (list -1 -2 -3)) true)

(define (andmap2 pred lst)
  (cond [(empty? lst) true]
        [else
         (and (pred (first lst))
              (andmap2 pred (rest lst)))]))
```

Multiple Choice

1/1 point (graded)

Which of the following is the correct signature for `andmap2`?

- `;; (X -> Y) (listof X) -> Boolean`
- **`;; (X -> Boolean) (listof X) -> Boolean`** ✔
- `;; (X -> Y) (listof Y) -> Y`
- `;; (Number -> Boolean) (listof Number) -> Boolean`

Explanation

The signature we get from `all-positive?` and `all-negative?` as well as the examples for `andmap2` is:

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

But we can make this function do more. We can for examplec all it with a `square?` predicate and a list of images to see if all the images are square. Then the signature would be:

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

So this leads us to an overall signature for `andmap2` of:

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

Submit Show Answer

Answers are displayed within the problem