Aleks Itskovich

Write a function that takes two lists of integers and returns a list containing tuples with corresponding elements from both the lists. For example - f ([1, 2, 3], [4, 5, 6]) -> [(1, 4), (1,5), (1,6), (2, 4), (2, 5), (2, 6), (3, 4), (3 ,5), (3, 6)]. If either list is null, the result is null. The lists do not have to be the same length. Solve this using recursion. You may **NOT** use the length() function or lambda() function or comprehension lists to do your solution.

Haskell

```haskell

permute :: [a] -> [b] -> [(a, b)]
permute [] _ = []
permute _ [] = []
permute (x:xs) (ys) = (x, head ys) : permute [x] (tail ys) ++ permute xs ys

```

Command Prompt - ghci

```
*Main> permute [1,2,3] [4,5,6]
[(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)]
*Main> permute [1] [4,5,6]
[(1,4),(1,5),(1,6)]
*Main> permute [1,2,3] [4]
[(1,4),(2,4),(3,4)]
*Main> permute [1,2,3] []
[]
*Main> permute [] [4,5,6]
[]
```

Scheme

```scheme


(define (permute_internal n lst)
    (cond
      ((null? lst) '())
      (else
            (append (list (cons n (list (car lst)))) (permute_internal n (cdr lst)))
)))

(define (permute lst1 lst2)
    (cond
        ((null? lst1) '())
        ((null? lst2) '())
        (else (append (permute_internal (car lst1) lst2) (permute (cdr lst1) lst2)))
    )
)

```

Or as a single function (if helpers are not allowed):

```scheme

(define (permute lst1 lst2)
    (cond
        ((null? lst1) '())
        ((null? lst2) '())
        (else
            (append
                (append
                        (list (cons (car lst1) (list (car lst2))))
                        (permute (list (car lst1)) (cdr lst2))
                )
                (permute (cdr lst1) lst2)
            )
        )
    )
)

```

```scheme
(newline)
(display (permute () '(4 5 6)))
(newline)
(display (permute '(1 2 3) '()))
(newline)
(display (permute '(1 2 3) '(4 5 6)))
(newline)
(display (permute '(1) '(4 5 6)))
(newline)
(display (permute '(1 2 3) '(4)))

```

```
$gosh main.sc

()
()
((1 4) (1 5) (1 6) (2 4) (2 5) (2 6) (3 4) (3 5) (3 6))
((1 4) (1 5) (1 6))
((1 4) (2 4) (3 4))
```