## Task 1:

For this task, we'll do some more practice with searching through multiple lists.

First, write a function `list_intersection(list1, list2)` that returns a list containing the elements that were found in **both** lists (note, this means an element must be present in both of the lists; an item present in just one list doesn't count).

Next, write a function, `list_sym_diff(list1, list2)` that returns a *symmetric differencing* of the two lists. A symmetric differencing is all of the elements that were present in one list, or the other list, but *not* in both lists.

Finally, write a function, `list_interleave(list1, list2)`. *Interleaving* two lists will grab the first element from the first list, then the first element from the second list, next the second element from the first list, and then the second element from the second list, and so on. **Note** that the lists do not necessarily have to be the same length.

The signature of all of these functions is `(list, list) ---> list`.

## Task 2:

Write a function, `third_index_of(list, item)` that finds the *third* index at which the specified item is found in the list. If the item was found in the list less than three times, return `None`.

The signature of this function is `(list, any) --> int`.

## Task 3:

Write a function `factors(n)` that returns the factors of an integer `n` as a `tuple`. For example `factors(12)` should return `(1,2,3,4,6,12)`.

The signature of this function is `(int) --> tuple(int, …)`

## Task 4:

Write a function `atleast_m_factors(n, m)` that returns the integers from 1 to `n` (inclusive) that have at least `m` factors as a `tuple`. For example `atleast_m_factors(1000,25)` should return `(720, 840, 900, 960)`. You'll probably want to use your `factors` function from the previous task to help with this.

The signature of this function is `(int, int) --> tuple(int, …)`