We smooth out some of the details of the linked list class.

1. Questions?

2. Recall: A linked list is: None if empty, otherwise, a head/tail pair.

3. Recall: Continuing steps to constructing a linked list:
   (a) Left off with: How do we append an element to the end of the list?
   (b) How do we get the nth element? (Useful for __getitem__.)
   (c) How do we reverse the list?
   (d) How do we know if two lists are ==?

4. The implementations of these methods are beautiful in their simplicity. However, they don't work on an empty list (because there is no object to work on!). Solution: a wrapper class.
   (a) A wrapper class (typically) protects the user from seeing the asymmetries associated with self-referential implementations of an empty structure.
   (b) We make the self-referential class private/hidden.
   (c) We build a separate public class that parallels the interface of the self-referential class.
      i. For example: The initializer creates an empty container.
      ii. For example: The append method either creates a list, or
      iii. calls append on the hidden list.
   (d) We hide the difficulties associated with the empty container object.
      i. For example: we introduce an empty method (as opposed to a standalone function) to test for an empty container.
      ii. For example: we can often move testing for an empty list into the wrapper.
   (e) Sentinel values: represent an empty list with a dummy element.

5. Ordered structures. What happens if you assume the structure is kept in order?
   (a) We can't have a method like append. Why?
   (b) We should avoid storing mutable values. Why?
   (c) We still need to be able to add and remove values.
   (d) The method __contains__ can make use of assumptions about order to run quickly.
   (e) These types of structures are very different than, say, Python's list class.

*