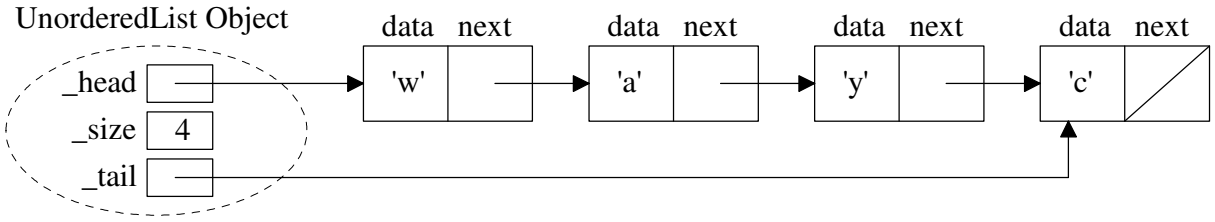
1. The textbook's unordered list ADT uses a singly-linked list implementation. I added the `_size` and `_tail` attributes:



a) The `search(targetItem)` method searches for `targetItem` in the list. It returns `True` if `targetItem` is in the list; otherwise it returns `False`. Complete the `search(targetItem)` method code:

```
class UnorderedList:

   def search(self, targetItem):
```

b) The textbook's unordered list ADT **does not** allow duplicate items, so operations `add(item)`, `append(item)`, and `insert(pos, item)` would have what precondition?

c) Complete the `append(item)` method including a check of it's precondition(s)?

```
   def append(self, item):
```

d) Why do you suppose I added a `_tail` attribute?

e) The textbook's `remove(item)` and `index(item)` operations "Assume the item is present in the list." Thus, they would have a precondition like "Item is in the list." When writing a program using an UnorderedList object (say `myGroceryList = UnorderedList()`), how would the programmer check if the precondition is satisfied?

```
itemToRemove = input("Enter the item to remove from the Grocery list: ")
if
   myGroceryList.remove(itemToRemove)
```

f) The `remove(item)` and `index(item)` methods both need to look for the `item`. What is inefficient in this whole process?

g) Modify the `search(targetItem)` method code in (a) to set additional data attributes to aid the implementation of the `remove(item)` and `index(item)` methods.

h) Write the `index(item)` method including a check of its precondition(s).

```
   def index(self, item):
```

i) Write the `remove(item)` method including a check of its precondition(s).

```
   def remove(self, item):
```