Linked lists are linear collections of nodes that contain data elements and a reference(also called link) to the next node in the list. Their order is not related to their location in memory, but by these references using pointers, the first one being the head. The last node(tail) is linked/pointed to "Null", which represents the end of the list.

In [1]:
```
# Take this list into account
basket = ['apples', 'grapes', 'pears']
```

linked list: apples --> grapes --> pears

- apples
  - 8947 --> grapes
    - 8742 --> pears
      - 372 --> null

Time complexities:

- prepend = O(1)
- append = O(1)
- lookup = O(n)
- insert = O(n)
- delete = O(n)

Linked lists makes appending and removing elements from any position of the list/sequence during an iteration without altering the whole list. However, to search and access these elements, it has to be done linearly from the beginning until the desired element/node(O(n)), whereas on a list/array you can access elements by using their index: O(1).

In Python, the way lists and linked lists manage memory is very similar but they differ in performance.

Even though inserting elements at the end of a list using .append() or .insert() will have constant time(O(1)), when you try inserting an element closer to or at the beginning of the list, the average time complexity will grow along with the size of the list: O(n).

On the other hand, linked lists are more straightforward when it comes to insertion and deletion of elements at the beginning or end of a list, where their time complexity is always constant: O(1).