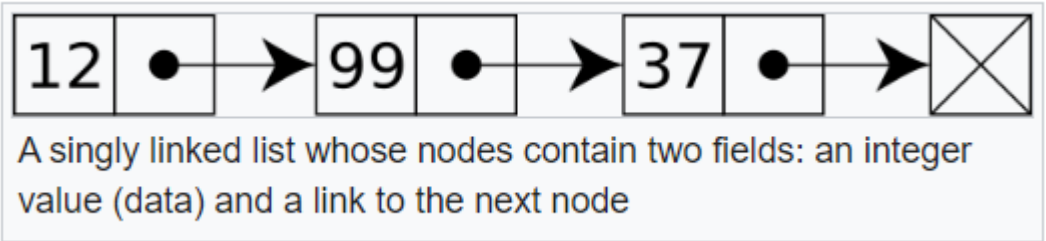
**Tuesday 15th October**

In a regular list, data is stored at successive locations in memory, and each data item has two components: its index in the list and the value of the data.

In contrast, a linked list is a sequence of nodes where each data item contains three fields: its index in the list and the value of the data, as well as a pointer (link) to the next node where the next data item in the list is stored in memory. Hence, the last node is linked to a terminator which is used to signify the end of the list.

For example:



A singly linked list whose nodes contain two fields: an integer value (data) and a link to the next node

The bullet point is an abstraction to the pointer to the memory location of the next data item in the linked list. The pointer of the last item is typically indicated by a pointer value of -1 or Null.

This means there are several advantages to linked lists over traditional lists:

| | **List** | **Linked List** |
|---|---|---|
| **Strengths** | - Fast search time<br>- Less memory needed per element | - Fast Insertion/Deletion time<br>- Dynamic size<br>- Efficient memory allocation/utilisation |
| **Drawbacks** | - Slow Insertion/Deletion time<br>- Fixed size<br>- Inefficient memory allocation/utilisation | - Slow search time<br>- More memory needed per node as additional storage is required for pointers |

Iterating (**traversing**) through a linked list:

- Check if the linked list is empty
- Start at the node the pointer is pointing to (Start = 1)
- Output the item at the current node (in this case >>> 12)

- Follow the pointer to the next node repeating through each node until the end of the linked list.
- When the pointer field is empty/null it signals that the end of the linked list has been reached.
- Traversing the above example would produce: >>> 12, 99, 37

Adding an item to the link list