Singly Linked Lists (SLL)

This lesson is a brief introduction to the functionality and purpose of the popular data structure called linked lists.

We'll cover the following

Introduction

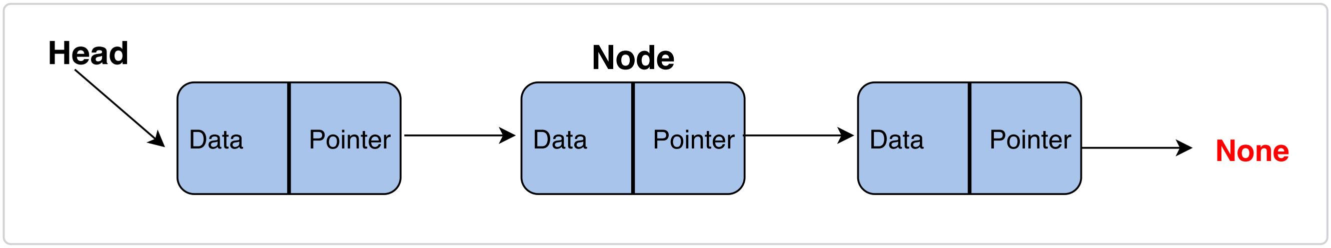
So far, we have seen how lists store and organize data for us. Python does not have a built-in linked list structure, as it isn't required after the introduction of lists.

However, knowledge about linked lists can be very useful in coding interviews! In this section, we will cover the basic behavior of linked lists and show how to implement them in Python syntax. This will also enhance your understanding on the back-end functionality of Python.

Let's get started!

Structure of A Singly Linked List

The most primitive form of the linked list data structure is th gl ke **list**. To start things off, let's take a look at its basic structure:



As you can see in the illustration above, a linked list is formed by **nodes** which are linked together like a chain.

Each node holds data along with a forward pointer to the next node in the list. The absence of a backwards pointer implies that there is a uni-directional link, i.e., the whole list points in one direction and cannot point backwards. This is why it is called a *singly* linked list.

Do not worry if you feel slightly lost at this point. For now, you only need to concern yourself with the two classes needed to implement a singly linked list:

- Node Class
- LinkedList Class

We'll discuss these one by one.

1. Node Class

The Node class has two components:

- **Data**
- **Pointer**

Data is the value you want to store in the node. Think of it as [illegible] a specific index in a list. The data type can range from *string* or *integer* to a user-defined class.

The **pointer** refers us to the next node in the list. It is essential for connectivity.

> Note that ‘pointer’ is a Node type variable.

Here’s a typical definition of a **Node** class:

Node.py

```
class Node:
    def __init__(self, data):
        self.data = data  # Data field
        self.next_element = None  # Pointer to next node
```

Explanation: As we discussed, the `Node` class has two variables. `data` contains your specified value and `nextElement` is the pointer to the next element of the list.

2. LinkedList Class#

The linked list itself is a collection of Node objects which we defined above. To keep track of the list, we need a pointer to the first node in the list.

This is where the principle of the **head** node comes in. The head does not contain any data and only points to the beginning of the list. This means that, for any operations on the list, we need to traverse it from the head (the start of the list) to reach our desired node in the list.

The implementation of the `LinkedList` class is shown below:

LinkedList.py

```
class LinkedList:
    def __init__(self):
        self.head_node = None  # Pointer to first node
```

Explanation: The implementation is fairly simple. When the list is first initialized it has no nodes, so the head is set to None.

Now that we've covered the basic implementation, let's discuss the differences between linked lists and lists.

Back
Lists Quiz: Test your understanding of ...

Next
Linked Lists vs. Lists

Completed

Report an Issue