List Data Structure

Vivek Singh

Information Systems Decision Sciences (ISDS)
MUMA College of Business
University of South Florida
Tampa, Florida

2018

Introduction to Lists

A list is a powerful yet simple data structure that helps us

- to build other complex data structures.
- create new data types and behaviors.
- store and retrieve data in a sequential way.

In simple terms, it is a collection of data elements ordered sequentially. Python provides us with a List collection type.

The Notion of List

- In the event that the language does not have a built in List type, the notion of a List has to be implemented.
- Recall that an abstract data type is only a Mathematical notion and it needs to be physically implemented to become a data structure.
- The behavior of the list can be easily implemented by defining functions.

There are two types of Lists

- An Unordered List
- An Ordered List

The Unordered List

- In an unordered list, each item in the collection holds a relative position with respect to other items.
- There is no emphasis on maintaining positioning of items in contiguous memory blocks.
- Hence there needs to be a mechanism to identify the location of each item in the memory.
- This can be realized by using a construct known as a Linked list.
- In a Linked list, each element holds a reference to the next element in the list.
- This is an explicit mechanism that helps each item identify the next item in the list.

- This helps us to maintain the relative positioning of the items which is the integral characteristic of an unordered list.
- Thus, each item can be accessed by simply following the links.
- The most important knowledge here is that the location of the first item in the list must be explicitly specified.
- The first element in the list is called the head and it holds the handle to accessing the rest of the content in the list.

The Node

A Node is the fundamental building block of a List.

- Each Node contains two essential pieces of information.
- First, it should contain the item itself which contains the data of that particular node.
- Second, the node contains a reference to the following node in the list.

This notion of a node can be implemented using a Class construct with attributes and methods as follows,

- data - contains the data stored in the item
- next - contains a reference to the next item.
- methods to set and get the data and the reference.

Python code

Code Implementation in Python

```
class Node:
        def __init__(self, init_data):
                self.data = init_data
                self.next = None
        def get_data(self):
                return self.data
        def get_next(self):
                return self.next
        def set_data(self, new_data):
                self.data = newdata
        def set_next(self, new_next):
                self.next = new_next
```

The Unordered List Class

The unordered List is constructed by using a Node as the fundamental building block. Each Node represents an item in the List.

- As previously stated, the reference to the first item in the list is the only way to access the entire list.
- The class definition should therefore contain an attribute to represent the first item, which we commonly refer to us "Head".

```
class UnorderedList:
        def __init__(self):
                self.head = None
```

Continued..

Notice that in the class definition, we used the special Python type "None".
The reasons and advantages to use "None" are many,

- The special reference None can be used to state that the head of the list does not refer to anything as yet.
- if the "next" attribute of a Node contains "None", it means there is no next node in the list.

Also it is important to understand that the list class itself does not contain any node objects.
It only contains a single reference to only the first node in the linked structure.

Essential Operations in a List

- Create List instances on the go.
- Ability to check if the list is empty.
- More importantly, the ability to add items to the list in an efficient manner.
- Ability to remove items from the list.
- Ability to search and find an item in the list.
- To find the number of items (size) in the list.

Implementing the Add Method

The most important feature of a list is the ability to add elements to it. And Python lists are not limited by the type of data that can be added to the list. It is versatile.
It could be implemented as follows,

- Since this is an unordered list, the position of the new item in the list in relation with other items is not important.
- The item only needs to be added to the list and become part of the linked structure.
- Thereby, we could think of many ways to add the element to the easiest position possible, which happens to be the head of the list.
- The advantage is obvious that the add operation will always be O(1).

Implementing the Add Method..

- Recall that each item in the List is a Node and the linked list provides only a single entry point to the entire structure.
- Hence, adding the new item to the head is the best possible approach.

It works in two important steps,

- The "next" reference of the new item(node) will point to the current "Head" of the list.
- The new item then replaces the existing "Head" item to become the new head of the list.

The order of these two steps is of extreme importance to protect the integrity of the structure.

```
def add(self, item):
        temp = Node(item)
        temp.set_next(self.head)
        self.head = temp
        for i in range(1,10):
                print(i)
```

Summary

- List is one of the most important built-in data structure in Python.
- Lists can be implemented using the mathematical notion of it, either as an Unordered List or an Ordered List.
- The implementation can be done using an Object-oriented approach with class definitions for the list operations while representing each item in the list as a node.
- The list type offers many advantages that it can be used to implement other abstract data structures.
- Choosing the right implementation, makes the list operations quick and efficient such as the add operation which is O(1).