Today We Will Cover:

- The List ADT
  - The types of things all lists can do
  - A sneaky implementation of our own list class

- Nodes
  - A precursor to making Linked Lists – another type of list object

List ADT

- A List ADT:
  - a *list* is a collection of items where each item holds a **relative position** with respect to the others. We can consider the list as having a first item, a second item, a third item, and so on. We can also refer to the **beginning** of the list (the first item) and the **end** of the list (the last item)
  - We can *add* and *remove* items from a list
  - We can *search* for the existence of an item in a list
  - We can make a distinction between *ordered* and *unordered* lists:

`54, 26, 93, 17, 77, 31`

Items are not stored in a sorted fashion

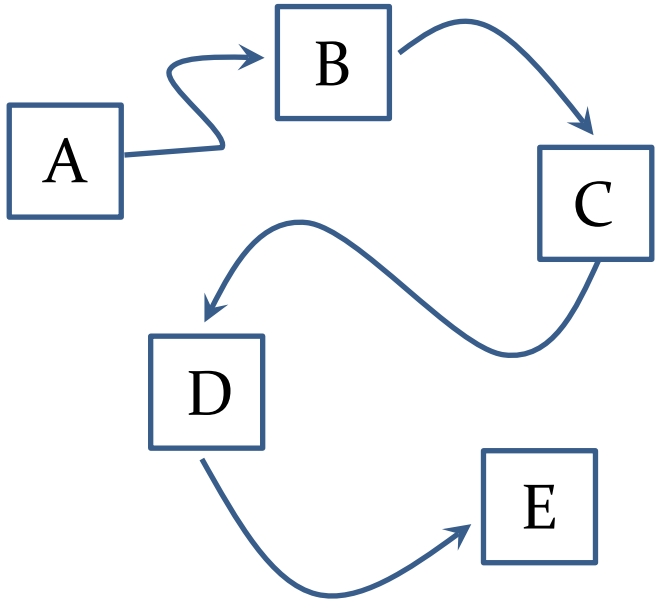
`17, 26, 31, 54, 77, 93`

Items are stored in a sorted fashion

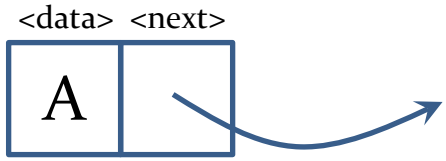
  - We will start by considering only unordered lists

Linked Lists

- Idea: rather than storing elements in consecutive memory locations, we can store them anywhere and link them together
- We do this using nodes, where each node contains one data element and a link to the next element.



A "node":



The actual data stored in the list item

The link to the next item in the list

- The **`Node`** is the basic building block of a linked list

Nodes

- What do nodes do?
  - Nodes remember some data
  - Nodes can have a reference to another node – often called ‘next’

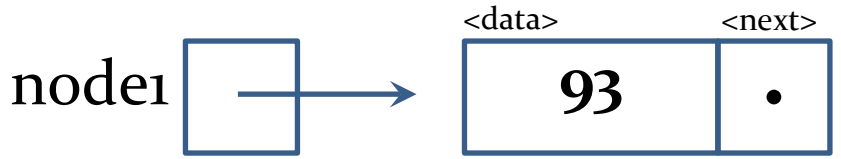
- When implementing a **`Node`** class we have getters and setters for the data and the next node
  - get_data(…), set_data(…)
  - get_next(…), set_next(…)

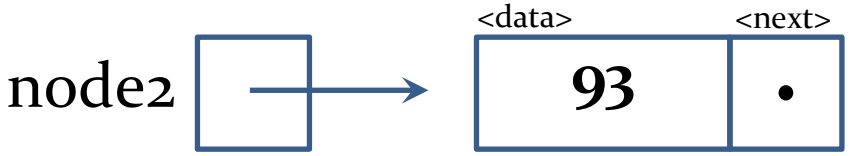
The Node class

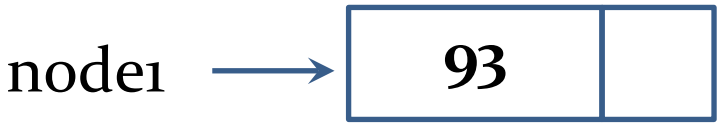
```
class Node:
    def __init__(self, init_data, next = None):
        self.__data = init_data
        self.__next = next

    def get_data(self):
        return self.__data

    def get_next(self):
        return self.__next

    def set_data(self, new_data):
        self.__data = new_data

    def set_next(self, new_next):
        self.__next = new_next
```

Example:

```
node1 = Node(93)
node2 = Node(93)
```



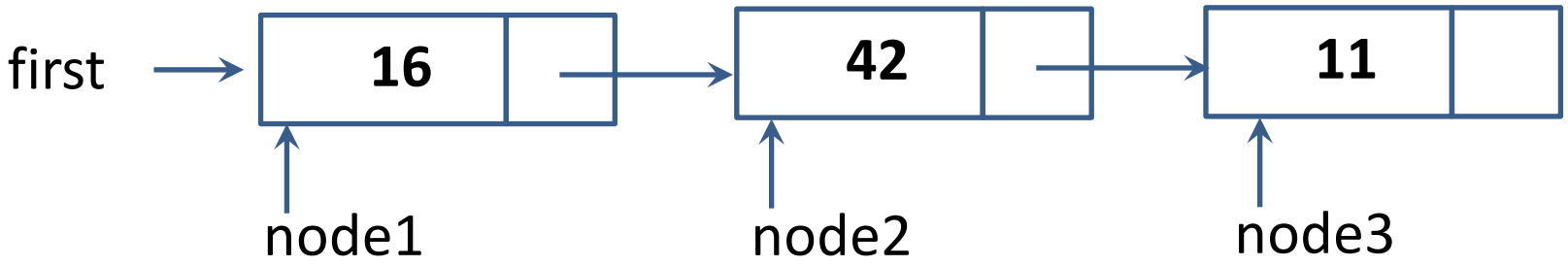




We often use the simpler notation:

Example

```
node1 = Node(16)
node2 = Node(42)
node3 = Node(11)
first = node1
node2.set_next(node3)
node1.set_next(node2)
```



What is the output of:

```
print(node3.get_data())
print(node1.get_next().get_data())
print(first.get_next().get_next().get_data())
```

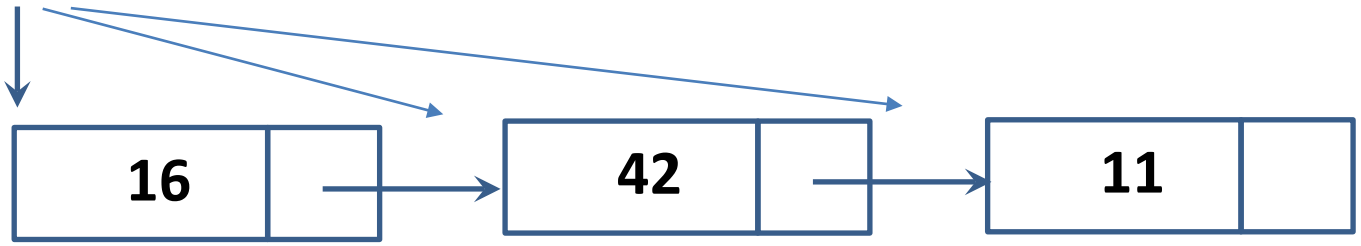
Methods

- Question 1 – 5:
  - add the xxxx method to the Node class

```
class Node:
    def __init__(self, init_data, next = None):
    ...

    def ...
```

  - you may find the following code fragment is useful:

```
current = self.__next
while current != None:
    …
    current = current.__next
```



Functions

- Question 6 – 10:
  - you will be defining a function which **USES** the Node ADT. A node implementation is provided. Your code can make use of any of the Node ADT methods: Node(), get_data(), set_data(), get_next(), set_next(), and get_sum().

```
def xxx(a_node):
    ...
```

  - you may find the following code fragment is useful:

```
current = xxx
while current != None:
    ...
    current = current.get_next()
```

use the get_next() method