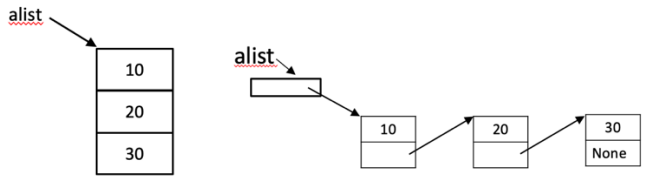
Work with your neighbor. (This will be graded for participation only.)

For reference, here are diagrams of a Python list and a `LinkedList` having the integers 10,20 and 30 as elements:



1. The `LinkedList` code includes a method `add()` for adding to the front of a `LinkedList`:

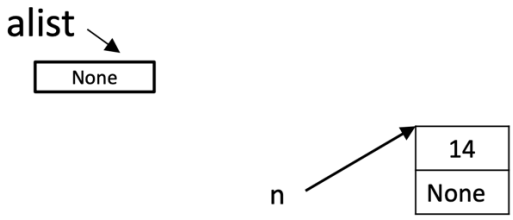
```
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None
```

```
class LinkedList:
    def __init__(self):
        self._head = None

    def add(self, new):
        new._next = self._head
        self._head = new
```

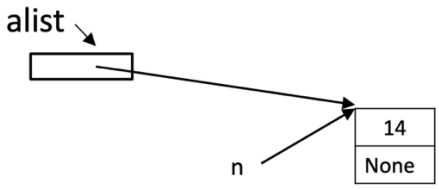
Draw a diagram that shows the linked list `alist` and the node `n` after the statements below are executed:

```
>>> alist = LinkedList()
>>> n = Node(14)
```



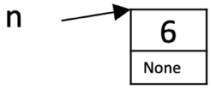
Draw a diagram that shows the list `alist` after the statement below is executed:

```
>>> alist.add(n)
```



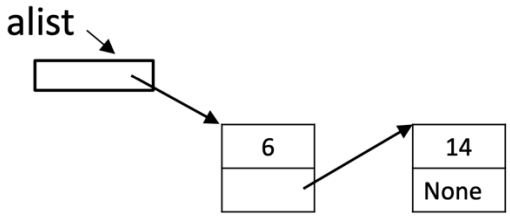
Draw a diagram that shows the node `n` after the statement below is executed:

```
>>> n = Node(6)
```

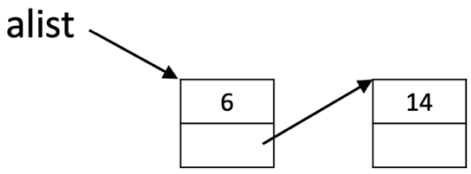


Draw a diagram that shows the list `alist` after the statement below is executed:

```
>>> alist.add(n)
```



Note: You may draw the diagram without the box for the list head and without labelling the last node's `_next` reference with `None`.



2. This code has a method for traversing a `LinkedList` to print all node values.

```
class Node:
    def __init__(self, value):
        self._value = value
        self._next = None
class LinkedList:
    def __init__(self):
        self._head = None

      def add(self, new):
        new._next = self._head
        self._head = new
```

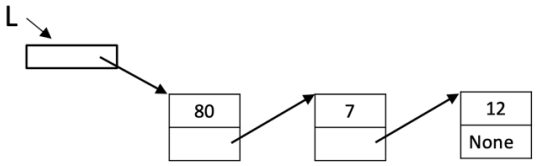
```
    def print_elements(self):
        current = self._head
        while current != None:
            print(str(current._value))
            current = current._next
```

Using the `LinkedList` and `Node` class definitions above, write a method `double()` for the LinkedList class that doubles the value attributes of all nodes in a `LinkedList`. You may assume that for each node in the list, the value attribute is an integer.

ANS:

```
    def double(self):
        current = self._head
        while current != None:
            current._value = current._value * 2
            current = current._next
```

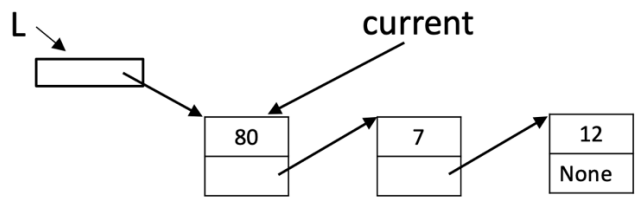
3. The code in Problem 2 has a method `print_elements(self)` for traversing a `LinkedList` to print all node values. Let's go through the steps to do that for the list below:



When the code `L.print_elements()` is called, `self` will refer the list `L`. The slides showed how to set a reference to the first element of the list, and continually change that reference to "walk" through the list. Using the variable `current`, what is the code to set `current` to the first element of the list? Show the code and draw the list with `current` set to the first element:

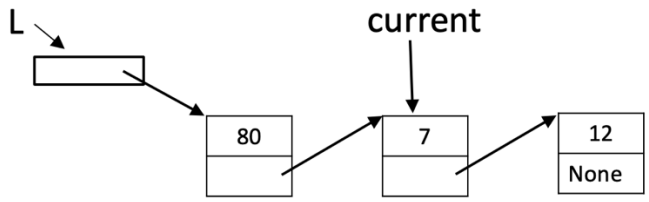
ANS: The code in lecture was written as a method; `self` **will reference the list** `L` **above.**

```
current = self._head
```



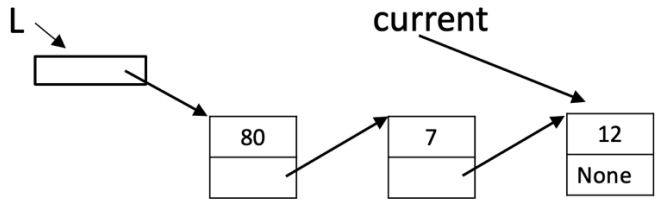
What is the code to move `current` forward to the next (second) node? Show the code and the resulting diagram after moving forward again:

```
current = current._next
```



Show the code to move `current` forward to the next (third) element and the diagram:

```
current = current._next
```



Show the code to move `current` forward one more time. What is the value of `current` is at this point?

```
current = current._next
```

`current` is now `None`

4. (Extra.) Using the `LinkedList` and `Node` class definitions above, write a method `print_evens(self)` for the `LinkedList` class that prints the even values of the nodes. If there are no even values, the method does not print anything. You can assume that all values in the nodes are integers.

ANS:

```
def print_evens(self):
    current = self._head
    while current != None:
        if current._value % 2 == 0:
            print(str(current._value))
        current = current._next
```