Singly Linked List vs Array

We are going to compare the timing for some operations on a linked list vs an array. Because of the way python is implemented this isn't exactly apples to apples but it will give us some idea. First you will need to implement the missing functions.

Some of the functions come from the were featured in the video from the last homework. You can just go back, watch the videos and copy those. Some of the other functions I added. You will need to look at the comments to understand what they do. There are two other sources to guide you.

1. In the cell after the one below, are cells that create a link list and exercise some of the functions. Use those to reason how the functions can work. Keep a printout or pdf of the original notebook so you know what the output is suppose to look like.
2. The cell with the unittests must not be changed. When you evaluate this cell it will test the functions in the class. Any failures are indicative of problems in your code. Do not change the unittest cell. You will lose points. Instead fix your code.

It is very rare that you should ever need numbers other than 0, 1 and sometimes 2 in production code. Sometimes there are parameters which are stored in variables and part of configuration. This code doesn't need that so the solution should not have any explicit constants in the code besides 0 or 1.

In [35]:
```
class LinkedNode:
    """
    Implements a single node in a linked list data structure.

    Attributes
    ----------
    value : object
    next: LinkedNode

    """
    def __init__(self, value, tail=None):
        """
        LinkedNode constructor

        Parameters
        ----------
        value : object
        next : LinkedNode, optional

        "value" attribute holds a reference to the data the node holds
        "next" attribute holds a reference to the next node in the lis
t.
        By default this is None indicated the node is the last in the
list.

        """
```

```
        self.value = value
        self.next = tail

class LinkedList:
    """
    Implements a single node in a linked list data structure.

    Attributes
    ----------
    head : LinkedNode
    count: int

    "head" attribute holds a reference to the first node in the linked
list.
    "count" holds the number of nodes (and thus values) in the link li
st.

    """
    def __init__(self,*start ):
        """
        LinkedList constructor

        Parameters
        ----------
        *start : object, multiple optional arguements

        "start" list of arguments used initialize the link list each b
eing used as a value.
        If empty the "head" attribute holds empty and the "count" attr
ibute set to 0.
        Each argument is **prepended** to the link list by first rever
sing the list
        of arguments to order is preserved.

        """
        self.head = None
        self.tail = None
        self.count = 0
        start=list(start)
        start.reverse()
        for _ in start:
            self.prepend(_)

    def prepend(self, value):
        """
        Add value to the front of the list. O(1)

        Parameters
        ----------
        value : object

        value arguement that should be store in a LinkNode at the fron
```

```
t of the list.
        Count increased by one.

        Returns
        -------
        None
        """

        self.head = LinkedNode(value,self.head)

    def __getitem__ (self, index):
        """
        Get the value at the index passed in. O(index)

        Parameters
        ----------
        index : integer

        Returns the values at index "index."

        Note: does not change value. Also note that
        slices not handled nor negative values.

        Returns
        -------
        object

        Raises
        ------
        Exception

        Exception raised if index out of range or not integer.
        Message states the index passed in and the size.
        """

        # Fill this in
        # if the index is out of range should raise an exception like
this
        # raise Exception("Index: {} out of range in link list of size
{}".format(index,current_index))
        current_index = index
        try __len__(self)>index:
            while index > -1:
                val = self.next
                index -= 1
        except:
            raise Exception("Index: {} out of range in link list of si
ze {}".format(index,current_index))

    def insert_value_at(self,value,index):
        """
        Insert a LinkNode at index, O(index)
```

```
        Parameters
        ----------
        index : object

        Returns the values at index "index."

        Note: does not change value. Also note that
        slices not handled nor negative values.

        Returns
        -------
        object

        Raises
        ------
        Exception

        Exception raised if index out of range or not integer.
        Message states the index passed in and the size.
        """

        # Fill this in
        # if the index is out of range should raise an exception like
this
        #    raise Exception("Index: {} out of range in link list of s
ize {}".format(index,current_index))

    def delete_at(self,index):
        """
        Delete a value at index and decrease the count by one, O(index
)

        Parameters
        ----------
        index : int


        Returns
        -------
        True if succesful

        Raises
        ------
        Exception

        Exception raised if index out of range or not integer.
        Message states the index passed in and the size.
        """
        # Fill this in
        # if the index is out of range should raise an exception like
this
        #    raise Exception("Index: {} out of range in link list of s
ize {}".format(index,current_index))
```

```
def remove_first(self,value):
    """
    Delete the first node with matching value, O(n)

    Parameters
    ----------
    value : object


    Returns
    -------
    True if succesful or false if value not found

    """
    # Called "remove" in the video
    n = self.head
    last = None
    while n != None:
        if n.value == value:
            if last == None:
                self.head = self.head.next
            else:
                last.next = n.value
            return True

        n = n.next
    return False



def pop(self):
    """
    Delete the first node and return value, O(1)


    Returns
    -------
    object

    Raises
    --------
    Exception('Nothing to Pop. List Empty')

    """
    # fill in
    # if list is empty raise this exception
    #    raise Exception('Nothing to Pop. List Empty')
    if self.head is None:
        raise Exception ("Empty List.")
    val = self.head.value
    self.head = self.head.next
    return val
```

```
    def __iter__(self):
        """
        interator for all values in the list, O(n)


        Yields
        -------
        object

        """
        # Fill in
        # Remember no return here only yeild
        n = self.head
        while n != None:
            yield n.value
            n = n.next

    def __len__(self):
        """
        returns the length of the series, O(1)


        Returns
        -------
        int
        """
        # Just relies on self.count for speed.
        return self.count

    def __repr__(self):
        """
        returns a string representation
        """
        if self.head is None:
            return "Link:[]"
        return 'LinkedList({})'.format(','.join(str(value) for value i
n self))
```

```
  File "<ipython-input-35-003650644f3f>", line 111
    try __len__(self)>index:
            ^
SyntaxError: invalid syntax
```

In [36]:
```
llist = LinkedList()
print(llist.count)
print(llist.head)
print(llist)
llist = LinkedList(0,88,2)
print(llist.count)
print(llist)
print(llist.head.value)
print(llist.head.next.value)
print(llist.head.next.next.value)
llist = LinkedList()
llist.prepend(1)
llist.prepend(3)
llist.prepend(5)
print(llist)
print(llist[2])
print(llist.insert_value_at(77,3))
print(llist)
print(len(llist))
llist.delete_at(1)
print(llist)
print(len(llist))
llist.remove_first(1)
print(llist)
print(llist.pop())
print(llist)
llist = LinkedList(0,88,2)
print(list(llist))
```

```
0
None
Link:[]
0
LinkedList(0,88,2)
0
88
2
LinkedList(5,3,1)
None
None
LinkedList(5,3,1)
0
LinkedList(5,3,1)
0
LinkedList(3,1)
3
LinkedList(1)
[0, 88, 2]
```

In [37]:
```
# DO NOT CHANGE ANYTHING in this CELL!!!
# This must run AS IS without error after evaluating your code
# you can see the error messages for information
```

```
import unittest
import doctest


class TestLinkList(unittest.TestCase):
    def test_empty_construct(self):
        self.llist = LinkedList()
        self.assertEqual(self.llist.count,0)
        self.assertEqual(self.llist.head,None)

    def test_single_construct(self):
        self.llist = LinkedList(88)
        self.assertEqual(self.llist.count,1)
        self.assertEqual(self.llist.head.value,88)
        self.assertEqual(self.llist.head.next,None)

    def test_multiple_construct(self):
        self.llist = LinkedList(0,88,2)
        self.assertEqual(self.llist.count,3)
        self.assertEqual(self.llist.head.value,0)
        self.assertEqual(self.llist.head.next.value,88)
        self.assertEqual(self.llist.head.next.next.value,2)

    def test_prepend(self):
        self.llist = LinkedList()
        self.assertEqual(self.llist.head, None)
        self.assertEqual(self.llist.count, 0)
        self.llist.prepend(88)
        self.assertEqual(self.llist.head.value,88)
        self.assertEqual(self.llist.count,1)
        self.llist.prepend(100)
        self.assertEqual(self.llist.head.value,100)
        self.assertEqual(self.llist.count,2)

    def test_getittem(self):
        self.llist = LinkedList(798,9,200)
        self.assertEqual(self.llist[0],798)
        self.assertEqual(self.llist[1],9)
        self.assertEqual(self.llist[2],200)
        with self.assertRaises(Exception):
            self.llist[3]

    def test_insert_value_at(self):
        self.llist = LinkedList(798,9,200)
        self.llist.insert_value_at(99,0)
        self.assertEqual(self.llist[0],99)
        self.assertEqual(self.llist[1],798)
        self.assertEqual(self.llist[2],9)
        self.assertEqual(self.llist[3],200)
        self.llist.insert_value_at(77,3)
        self.assertEqual(self.llist[1],798)
        self.assertEqual(self.llist[2],9)
        self.assertEqual(self.llist[3],77)
```

```
        self.assertEqual(self.llist[4],200)
        self.llist.insert_value_at(8,5)
        self.assertEqual(self.llist[1],798)
        self.assertEqual(self.llist[2],9)
        self.assertEqual(self.llist[3],77)
        self.assertEqual(self.llist[4],200)
        self.assertEqual(self.llist[5],8)
        with self.assertRaises(Exception):
            self.llist.insert_value_at(2,7)

    def test_delete_at(self):
        self.llist = LinkedList(798,9,200)
        self.assertEqual(self.llist.delete_at(1),True)
        self.assertEqual(self.llist.count,2)
        self.assertEqual(self.llist[0],798)
        self.assertEqual(self.llist[1],200)
        with self.assertRaises(Exception):
            self.llist.delete_at(2)

    def test_remove_first(self):
        self.llist = LinkedList(798,9,200)
        self.assertEqual(self.llist.remove_first(200),True)
        self.assertEqual(self.llist.count,2)
        self.assertEqual(self.llist[1],9)
        self.assertEqual(self.llist.remove_first(10),False)
        self.assertEqual(self.llist.count,2)

    def test_pop(self):
        self.llist = LinkedList(798,9,200)
        self.assertEqual(self.llist.pop(),798)
        self.assertEqual(list(self.llist),[9,200])
        self.assertEqual(self.llist.pop(),9)
        self.assertEqual(list(self.llist),[200])
        self.assertEqual(self.llist.pop(),200)
        self.assertEqual(list(self.llist),[])
        with self.assertRaises(Exception):
            self.llist.pop()

    def test_iter(self):
        self.llist = LinkedList()
        self.assertEqual(list(self.llist),[])
        self.llist = LinkedList(67,88,42)
        self.assertEqual(list(self.llist),[67,88,42])

    def test_len(self):
        self.llist = LinkedList()
        self.assertEqual(len(self.llist),0)
        self.llist = LinkedList(8)
        self.assertEqual(len(self.llist),1)
        self.llist = LinkedList(7,15)
        self.assertEqual(len(self.llist),2)
        self.llist = LinkedList(15,37,43,51)
        self.assertEqual(len(self.llist),4)
```

```
    def test_repr(self):
        self.llist = LinkedList(15,37,43,51)
        self.assertEqual(str(self.llist),'LinkedList(15,37,43,51)')


unittest.main(argv=[''], verbosity=2, exit=False)
```

```
test_delete_at (__main__.TestLinkList) ... FAIL
test_empty_construct (__main__.TestLinkList) ... ok
test_getittem (__main__.TestLinkList) ... FAIL
test_insert_value_at (__main__.TestLinkList) ... FAIL
test_iter (__main__.TestLinkList) ... ok
test_len (__main__.TestLinkList) ... FAIL
test_multiple_construct (__main__.TestLinkList) ... FAIL
test_pop (__main__.TestLinkList) ... ok
test_prepend (__main__.TestLinkList) ... FAIL
test_remove_first (__main__.TestLinkList) ... FAIL
test_repr (__main__.TestLinkList) ... ok
test_single_construct (__main__.TestLinkList) ... FAIL

======================================================================
==
FAIL: test_delete_at (__main__.TestLinkList)
----------------------------------------------------------------------
--
Traceback (most recent call last):
  File "<ipython-input-37-cef3d3d8c186>", line 69, in test_delete_at
    self.assertEqual(self.llist.delete_at(1),True)
AssertionError: None != True

======================================================================
==
FAIL: test_getittem (__main__.TestLinkList)
----------------------------------------------------------------------
--
Traceback (most recent call last):
  File "<ipython-input-37-cef3d3d8c186>", line 40, in test_getittem
    self.assertEqual(self.llist[0],798)
AssertionError: None != 798

======================================================================
==
FAIL: test_insert_value_at (__main__.TestLinkList)
----------------------------------------------------------------------
--
Traceback (most recent call last):
  File "<ipython-input-37-cef3d3d8c186>", line 49, in test_insert_va
lue_at
    self.assertEqual(self.llist[0],99)
AssertionError: None != 99

======================================================================
==
```

```
FAIL: test_len (__main__.TestLinkList)
----------------------------------------------------------------------
--
Traceback (most recent call last):
  File "<ipython-input-37-cef3d3d8c186>", line 105, in test_len
    self.assertEqual(len(self.llist),1)
AssertionError: 0 != 1

======================================================================
==
FAIL: test_multiple_construct (__main__.TestLinkList)
----------------------------------------------------------------------
--
Traceback (most recent call last):
  File "<ipython-input-37-cef3d3d8c186>", line 22, in test_multiple_
construct
    self.assertEqual(self.llist.count,3)
AssertionError: 0 != 3

======================================================================
==
FAIL: test_prepend (__main__.TestLinkList)
----------------------------------------------------------------------
--
Traceback (most recent call last):
  File "<ipython-input-37-cef3d3d8c186>", line 33, in test_prepend
    self.assertEqual(self.llist.count,1)
AssertionError: 0 != 1

======================================================================
==
FAIL: test_remove_first (__main__.TestLinkList)
----------------------------------------------------------------------
--
Traceback (most recent call last):
  File "<ipython-input-37-cef3d3d8c186>", line 79, in test_remove_fi
rst
    self.assertEqual(self.llist.count,2)
AssertionError: 0 != 2

======================================================================
==
FAIL: test_single_construct (__main__.TestLinkList)
----------------------------------------------------------------------
--
Traceback (most recent call last):
  File "<ipython-input-37-cef3d3d8c186>", line 16, in test_single_co
nstruct
    self.assertEqual(self.llist.count,1)
AssertionError: 0 != 1

----------------------------------------------------------------------
--
```

```
Ran 12 tests in 0.029s

FAILED (failures=8)
```

Out[37]: `<unittest.main.TestProgram at 0x105990be0>`

Test Output

Running the above test should give output like this:

```
test_delete_at (__main__.TestLinkList) ... ok
test_empty_construct (__main__.TestLinkList) ... ok
test_getittem (__main__.TestLinkList) ... ok
test_insert_value_at (__main__.TestLinkList) ... ok
test_iter (__main__.TestLinkList) ... ok
test_len (__main__.TestLinkList) ... ok
test_multiple_construct (__main__.TestLinkList) ... ok
test_pop (__main__.TestLinkList) ... ok
test_prepend (__main__.TestLinkList) ... ok
test_remove_first (__main__.TestLinkList) ... ok
test_repr (__main__.TestLinkList) ... ok
test_single_construct (__main__.TestLinkList) ... ok

----------------------------------------------------------------------
Ran 12 tests in 0.012s

OK
```

Timing

Here you should test to see which code is faster. Prepending a link list or prepending (inserting at the beginning) of a python array. Note the %%timeit magic command is good for benchmarking.

In [31]:
```
# Number of inserts
num_prepends = 100000
```

In [32]:
```
%%timeit
new_llist = LinkedList() #small initialization
for ind in range(num_prepends):
    new_llist.prepend(9)
```

```
342 ms ± 52.3 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [33]:
```
%%timeit
reg_list = [] #small initialization
for ind in range(num_prepends):
    reg_list.insert(0, 9)
```

```
5.21 s ± 3.07 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

In [ ]: