Name:_____ **Partner:** _____

Python Activity 43: Linked Lists – Wrapper Class

Learning Objectives
Students will be able to:
Content:
- Define a **wrapper class**
- Explain the shortcomings of a solitary **Element** class

Process:
- Write code that adds and modifies elements of a LinkedList
- Write code to access elements of a LinkedList
- Write code to make an object iterable.

Prior Knowledge
- Python concepts from Activities 1-19, recursion, special methods, Linked Lists - Elements

Folks, this is a brand new activity. If you encounter any issues/typos, please let Iris know!

Critical Thinking Questions:

FYI: Continuing on from Linked Lists – Elements, we're building a ***Linked List*** data structure which is a series of ***Elements*** linked together, one pointing to the next. Review Activity 42 first!

1. Below are the special methods, `__init__(..)` and `__len__(..)` for an `Element` object.

In the Element Class

```
0 def __init__(self, value, next=None):
1         self._next = next
2         self._value = value
3 def __len__(self):
4         if self.next is None:
5             return 1
6         else:
7             return 1 + len(self.next)
```

```
8 >>> ee = Element(3)
```

   a. How many parameters does initializing a new `Element` require? _____

   b. At line 1, what is stored in `ee.next`? _____

   c. At line 2, what is stored in `ee.value`? _____

   d. If we added the following code at line 9, what would be returned?: `len(ee)`

   _____

   e. Is it possible to write a line of code to create an empty `Element` list (such that the length would be 0)? How?:

   _____

FYI: Because an `Element` list is constructed with `value` as a positional argument, we cannot construct an empty `Element` list!

2. The following code uses the `LinkedList` class in interactive python (indicated by >>>):

```
0 >>> ll = LinkedList()
1 >>> ll._head
2 None
3 >>> ll.append(3)
4 >>> ll._head.value
5 3
6 >>> type(ll._head)
7 <class 'Element'>
```

a. If we replaced line 1 with `len(ll)`, what should be returned? _____

b. If we replaced line 4 with `len(ll)`, what should be returned? _____

c. What is stored in `ll._head` at line 0? Where does that change?
Line 0 :_____ `ll._head` changes: _____

d. What type of object is `LinkedList._head`? _____

e. What might `LinkedList._head` represent?
_____

3. The following code is the `__len__(..)` method from `LinkedList`:

In the LinkedList Class

```
0 def __len__(self):
1     if self._head is None:
2         return 0
3     else:
4         return len(self._head)
```

a. Write a line of code that *implicitly* calls `__len__(..)` on our LinkedList, `ll`:
_____

b. If we constructed a new LinkedList, `ll = LinkedList()`, what would `__len__(..)` return?
_____

c. `len(..)` is being called on `self._head` on line 4. What class defines that `len(..)` method?
_____

d. If we appended a value to `ll`, as in the previous example, what would this method return?
_____

FYI: A ***wrapper class*** is any class which wraps/encapsulates the functionality of another class or component. In this case, LinkedList is a wrapper class which encapsulates the container class, Element.

4. Examine the following example method from the `LinkedList` class:

```
0 def mystery(self, a):
1     if self._head is None:
2         self._head = Element(a)
3     else:
4         self._head.mystery(a)
```

a. What does the following line 1 do?: `if self._head is None:`

b. What happens `if self._head is None` (line 2)?

c. What type of object is `self._head`? (*Hint*: see line 2)

d. When `self._head.mystery(a)` is invoked, what class's `mystery` method is being called?

e. If we called this new mystery method on an empty LinkedList with `ll.mystery(55)`, what would happen?

f. If we replaced line 4 with `self._head.append(a)` and called this method on a LinkedList which already contains the values 1, 2, 3, 4, with `ll.mystery(55)`, what would happen?

g. What should the `mystery` method be renamed to?

FYI: `__setitem__(self, i, v)` is a special method in python that is called when assigning a value to an indexed item on the left-hand side of an assignment operator.

5. In examining this code, the method on the top is called when line 4 is evaluated:

```
0 def __setitem__(self, i, v):
1     self.head[i] = v

2 >>> ll = LinkedList()
3 >>> ll.append(3)
4 >>> ll[0] = 55
5 >>> print(ll)
6 [55]
```

a. What is stored in `ll` after line 2? _____

b. What might happen if we executed `None.__setitem__(0, 55)`?

c. What might happen if we were to replace line 3 with `ll[0] = 77`?

d. What is stored in `ll` after the original line 3 above? _____

e. Write some code to add a second element to our list, `ll`, to have the value 99, and then change it to 33 using the `__setitem__` special method, implicitly:

Application Questions: Use the Python Interpreter to check your work

1. Write the `__str__(self)` method for our LinkedList class so that it prints the values of our list, just like when we print a python list (remember the square brackets and commas):

```
def __str__(self):
```

2. Write the `extend(self, v)` method for our LinkedList class so that it adds all the objects stored in `v` to the end of our list. Make use of the Element.append(self, v) method to avoid code redundancy:

```
def extend(self, v):
```

3. Write a method of `LinkedList` that returns True if the given value, `v`, exists as a value within the list, `False` if not contained in the `LinkedList`. Refer to the `LinkedList.__len__(self)` code in this activity for some hints on the approach and structure.

```
def __contains__(self, v):
```

4. Write a method, `__iter__(self),` for the LinkedList class that *yields* one item from our list at a time. This method would be called with a line like "`for item in ll:`". Refer to the activity on generators for more insight.

```
def __iter__(self):
```

_____

_____

_____

_____

_____

_____

_____

FYI: `__iter__(self)` is a special method in python that is called when using a for loop over that sequence object. Any class that has this method defined is ***iterable***.

5. Refer to Homework and Lab assignments for more application questions!