Name: ___________________  Partner: ___________________

Python Activity 31: 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, 28, Activity 30: 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 30 first!

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

In the Element Class

```python
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 positional arguments does initializing a new Element require? ______
b. How many keyword arguments does initializing a new Element require? ______
c. At line 5, what is stored in ee.next? ______________
d. At line 5, what is stored in ee.value? ______________
e. If we added the following code at line 9, what would be returned?: len(ee)

f. Write a line of code to create an empty Element list:

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:

```python
def mystery(self, a):
    if self._head is None:
        self._head = Element(a)
    else:
        self._head.mystery(a)
```

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

b. For this recursive method, what is the base case / stopping condition?

c. What is the small step we take in mystery for each recursive call?

d. For this recursive method, how is the longer journey broken down/shortened?

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

f. 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.

5. In examining this code, the method on the top is called when line 3 is evaluated:

```python
>>> ll = LinkedList()
>>> ll.append(3)
```

```python
55
```

b. If we typed `print(ll)` after line 2, what would be the output?

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

d. What might happen if we replaced line 3 with `ll[0] = 77`?

e. If we typed `print(ll)` after line 3, what would be the output?
f. Write some code to set the value of the second element of our list, 11, to 99, and then change it to 33:


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):

```python
def __str__(self):
    pass
```

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:

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

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.

```python
def __contains__(self, v):
    pass
```
4. Write a method, \texttt{\_\_iter\_\_(self)}, for the LinkedList class that \textit{yields} one item from our list at a time. This method would be called with a line like \texttt{\textit{for \textit{item in ll}:}}. Refer to the activity on generators for more insight.

\begin{verbatim}
def \_\_iter\_(self):
    
    
    
    
    
    
    
    
\end{verbatim}

\begin{tabular}{ |p{10cm}| }
\hline
\textbf{FYI:} \texttt{\_\_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 \textit{iterable}. \\
\hline
\end{tabular}

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