Critical Thinking Questions:

**FYI**: We’ve encountered python lists before, but now we’re going to build our own Linked Lists which are a series of Elements linked together, one pointing to the next.

1. The diagram below represents the underlying class structure for the list on the left.

```
ll = [3, 7, 1715]
```

<table>
<thead>
<tr>
<th>A Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element: _value 3</td>
</tr>
<tr>
<td>_next</td>
</tr>
<tr>
<td>Element: _value 7</td>
</tr>
<tr>
<td>_next</td>
</tr>
<tr>
<td>Element: _value 1715</td>
</tr>
<tr>
<td>_next</td>
</tr>
</tbody>
</table>

a. What are the two __slots__ of the Element class? ____________________________

b. What is stored in the _value variable of the first Element of this list? ____________

c. What is stored in the _next variable of the first Element of this list?

_______________________________________________________________________

d. What is stored in the _next variable of the last Element of this list?

_______________________________________________________________________

e. What does the _next variable represent?

_______________________________________________________________________
2. The following code creates a linked Element version of our list:

```python
yr = Element(1715)
d = Element(7, yr)
ll1 = Element(3, d)
```

a. What does the first parameter of a new `Element` instance represent?

b. What does the second parameter of a new `Element` instance represent?

c. Write a line of code to add “today” to the end of the ll1 list.

3. The following code creates another linked Element list:

```python
ll2 = Element(3)
ll2._next = Element(7)
ll2._next._next = Element(1715)
```

a. How does ll2 differ from ll1?

b. What would happen if we replaced `Element(1715)` with ll2 in the code above?

c. Write a function that allows `_next` to be set to a given Element. *(Hint: make use of a @next.setter annotation)*

e. Write an additional line that would ensure `_next` is an Element instance when set.

4. Examine the following example code:

```python
def mystery(self):
    if self.next is None:
        return 1
    else:
        return 1 + self.next.mystery()
```
b. What does the following line do?: `if self.next is None:`

________________________________________________________________________

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

________________________________________________________________________

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

________________________________________________________________________

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

________________________________________________________________________

f. For our example list, what will this `mystery` method return?

________________________________________________________________________

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

________________________________________________________________________

h. How can we construct an empty Element list?

________________________________________________________________________

FYI: `__getitem__(self, i)` is a special method in python that is called when accessing an indexed item.

5. In examining this code, the method on the right is called when the code on the left is evaluated:

```python
>>> ll[1]  # `ll` is a list
7
def __getitem__(self, i):
    if i == 0:
        return self.value
    else:
        return self.next[i-1]
```

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

________________________________________________________________________

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

________________________________________________________________________

f. What is the small step we take in `__getitem__` for each recursive call?
Application Questions: Use the Python Interpreter to check your work

1. Write the `__str__(self)` method for our `Element` class so that it prints the values of all the elements in our list, not just our first Element’s value:
   ```python
def __str__(self):
    # Your implementation here
    # Here's a possible implementation:
    # print all elements
    for element in self:
        print(element)
```  

2. Write the `append(self, v)` method recursively for our `Element` class so that it adds the object, `v`, to the end of our `Element` list. When considering the recursion, determine (1) what is the stopping condition, (2) what is the small step we should take with each recursive call, and (3) how do we break the journey down into a smaller journey:
   ```python
def append(self, v):
    # Your implementation here
    # Here's a possible implementation:
    # Base case: if the list is empty, add the new value
    if not self:
        self.append(v)
    # Recursive case: append the new value to the last element
    else:
        self[-1].append(v)
```  

3. Write a recursive method of `Element` that returns `True` if the given value, `v`, exists as a value within the list, `False` if not contained in the `Element` list.
   ```python
def __contains__(self, v):
    # Your implementation here
    # Here's a possible implementation:
    # Base case: if the list is empty, return False
    if not self:
        return False
    # Recursive case: check if the value is in the current element or its elements
    # If not, return False
    for element in self:
        if v in element:
            return True
        else:
            return False
```