# Python Activity 42: Linked Lists - Elements

Name: __________________________  Partner: __________________________

## Learning Objectives
Students will be able to:

**Content:**
- Define a **linked list**
- Identify the **value** and **next** of a linked list
- Explain the shortcomings of a solitary **Element** class

**Process:**
- Write code that adds elements to a list
- Write code that iterates through the list’s values.

**Prior Knowledge**
- Python concepts from Activities 1-19, Lists, Recursion

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

## 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 11 list on the left.

   ![A Linked List Diagram](image)

   \[11 = [3, 7, 1715]\]

   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:
   
   ```
   yr = Element(1715, None)
   d = Element(7, yr)
   l11 = 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 'founded' to the beginning of the 111 list.

   ____________________________________________________

   d. How might we construct an empty Element list?

   ____________________________________________________

3. The following code creates another linked Element list:
   
   ```
   l12 = Element(3)
   l12._next = Element(7)
   l12._next._next = Element(1715)
   ```

   a. How does l12 differ from 111?

   ____________________________________________________

   b. What would happen if we replaced `Element(1715)` with l12 in the code above?

   ____________________________________________________

   c. Write a line of code that would add 'in Williamstown' as the last element of l12.

   ____________________________________________________

4. Examine the following example code:

   ```
   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:`

   ____________________________________________________

   a. 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, ll1, what will this mystery method return?

g. What should the mystery method be renamed to?

### 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
```
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
```

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
```