Name:________________________________ **Partner:** ________________________________

Python Activity 42: Linked Lists - Elements

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

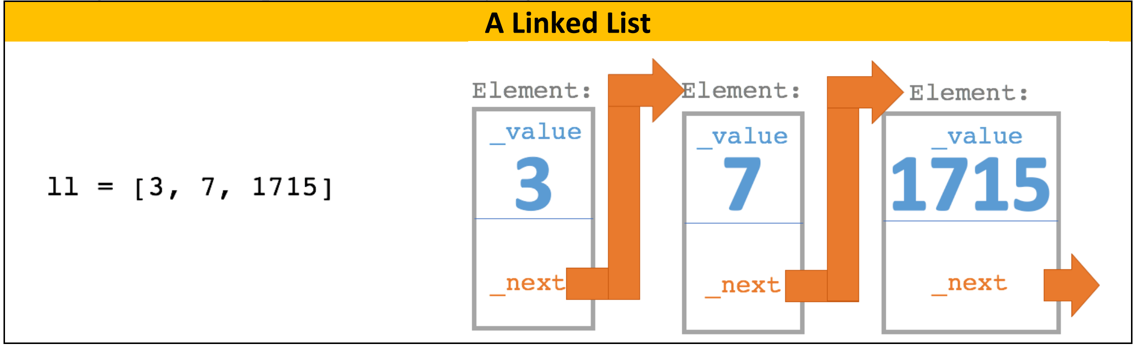
- Python concepts from Activities 1-19, Lists, Recursion

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 `ll` list on the left.



```
ll = [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)
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 `'founded'` to the beginning of the `ll1` list.

d. How might we construct an empty Element list?

3. The following code creates another linked Element list:

```
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 line of code that would add `'in Williamstown'` as the last element of `ll2`.

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?

FYI: `__getitem__(self, i)` is a special method in python that is called when accessing an indexed item. `i` is the index of the sequence being accessed.

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

```
>>> ll[1]          def __getitem__(self, i):
7                      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:

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

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:

```
def append(self, 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.

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