Name:_____ **Partner:** _____

Python Activity 44: Linked Lists - Methods

Digging deeper into the implementation of our own [recursive] list data structure.

Learning Objectives
Students will be able to:
Content:
- Compare & contrast **special methods** and dot-notation methods
- Explain how different modifications to a recursive list impacts efficiency

Process:
- Write code that modifies a recursive list class in multiple ways

Prior Knowledge
- Python concepts: Linked Lists, Recursion, User-defined types, Special methods

Concept Model:
CM1. We've been building out the *special methods* for our recursive `LinkedList` class, but not all important list methods are special methods!

a. What are some methods we use with Python `lists` that we call using dot notation?

_____

b. What is the difference between the methods from (a) and the *special methods* we've implemented previously?

_____

Critical Thinking Questions:
1. We want to write a recursive `append(self, val)` method for our `LinkedList` class that will append the value, `val`, to the end of our `LinkedList`:

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

   _____

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

   _____

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

   _____

   d. How might we actually *append* `val` to the end of our list?

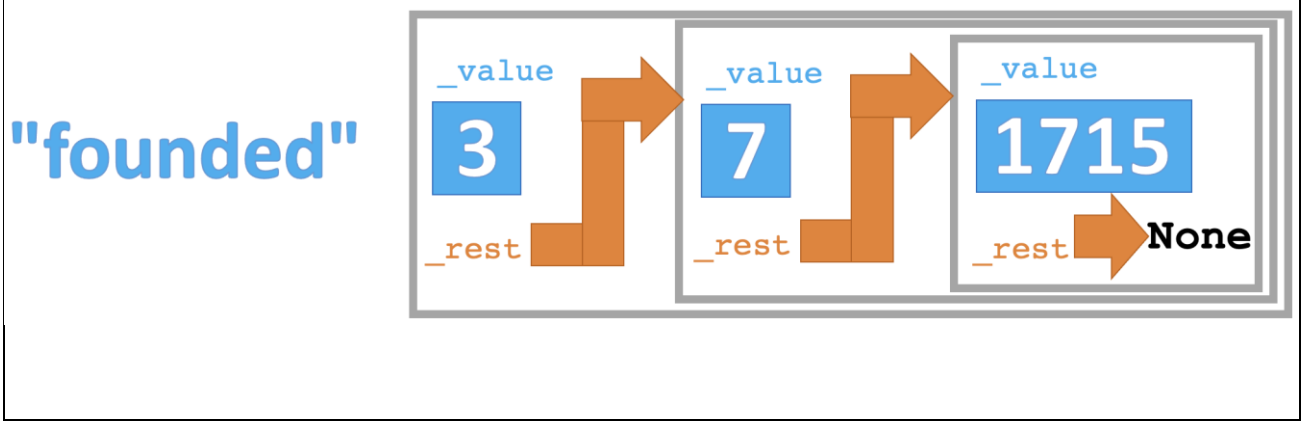
   _____

   e. Below is the completed implementation of the `append` method. Place a star next to the base case. Circle the recursive call. Underline where we actually append `val`. Does any of the code here surprise you?

```
def append(self, val):
    if self._rest is None:
        self._rest = LinkedList(val)
    else:
        self._rest.append(val)
```

2. We want to write a `prepend(self, val)` method for our `LinkedList` class that will place the value, `val`, at the *beginning* of our `LinkedList`. Below is a diagram of a sample `LinkedList` and a val, `"founded"`, to *prepend* to our list:

Linked List Diagram



a. Modify the diagram above to add the string `"founded"` to the beginning of the list.

What is the *new* `_value` of the 0th element of our list? _____
What is the *new* `_rest` of the 0th element of our list? _____

b. Do we need recursion to implement this method? Why/not?

_____

c. *Describe* your approach for implementing the `prepend` method:

_____
_____

d. Below is the completed implementation of the `prepend` method. Does any of the code here surprise you? Why/not?

_____

```
def prepend(self, val):
    old_val = self._value
    old_rest = self._rest
    self._value = val
    self._rest = LinkedList(old_val, old_rest)
```

3. We want to write an `insert(self, val, index)` method for our `LinkedList` class that will place the value, `val`, at the *index*, `index`, of our `LinkedList`:

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

   _____

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

   _____

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

   _____

   d. Below is the partially completed implementation of the `insert` method. Fill in the lines below the `(i)`, `(ii)`, and `(iii)` comments with Python code.

```
def insert(self, val, index):
    # (i) if index is 0, we add to beginning
    if


    # (ii) we've reached end of list, so append to end
    elif


    # (iii) else recurse until index reaches 0
    else:
```

Application Questions: Use the Python Interpreter to check your work

1. Write the `extend(self, other_lst)` method for our `LinkedList` class so that we can add a `LinkedList` to the end of the calling instance. 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 extend(self, other_lst):
```

_____

_____

_____

_____

_____

_____

2. Write a recursive method of `LinkedList` that returns a copy of the calling instance.

```
def copy(self):
```

_____

_____

_____

_____

_____

_____

3. Write a recursive `LinkedList` method that changes the values of the calling instance to `None`.

```
def clear(self):
```

_____

_____

_____

_____

_____

_____