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

Name:_____ **Partner:** _____

Python Activity 45: Iterators

Iterating over lists is useful, our LinkedList should be iterable as well!

Learning Objectives
Students will be able to:
Content:
- Define an **iterable**
- Summarize how an iterable works

Process:
- Write code that enables a user-defined class to be iterable
- Write code to iterate over a LinkedList

Prior Knowledge
- Python concepts: LinkedList, iteration

Concept Model:
We are building on our ***Linked List*** user-defined types with some added functionality – we want to *iterate* over our `LinkedList` objects!

CM1. Circle the built-in Python data types below that we can *iterate* over:

`bool`	`float`	`None`	`range`	`str`	`tuple`
`dict`	`int`	`list`	`set`	`TextIOWrapper` (a file)	

What does *iterate* mean?

_____

CM2. The following code on the left iterates over a `LinkedList` with a for..loop, and the code on the right *attempts* to use a for..each loop to iterate :

Iterating Over a Linked List

```
char_lst = LinkedList('a', LinkedList('b', LinkedList('c')))

for i in range(len(char_lst)):
    print(char_lst[i])
```

```
for item in char_lst:
    print(item)

TypeError: 'NoneType' object is
not subscriptable
```

a. What might be displayed by the code on the left?

_____

b. What `LinkedList` method is being called on the left?

_____

c. Why might the code on the right throw an error, but not on the left?

_____

FYI: To be an ***iterable***, a class has to implement the special methods, `__iter__(self)` and `__next__(self)`.

Critical Thinking Questions:

1. The following code assumes we have implemented the special methods, `__iter__(self)` and `__next__(self)` for our `LinkedList`:

Interactive Python

```
>>> char_lst = LinkedList('a', LinkedList('b', LinkedList('c')))
>>> list_iterator = iter(char_lst)
>>> next(list_iterator)
a
>>> next(list_iterator)
b
>>> next(list_iterator)
c
>>> next(list_iterator)
StopIteration ----> 1 next(list_iterator)
```

a. What special method might the built-in function `iter(..)` call? _____
What special method might the built-in function `next(..)` call? _____

b. How does the output from the first 3 calls to `next(list_iterator)` relate to the values in `char_lst`?

_____

What might the `next()` built-in function do?

_____

c. How many values are in `char_lst`? _____
What happens in the code above when we try to access more than this many values?

_____

How do we know when we've run out of elements to iterate over in an iterable?

_____

2. The following code extends our implementation of the `LinkedList` class with the special methods, `__iter__(self)` and `__next__(self)` such that the behavior in the previous question is implemented:

linkedlist.py

```
def __iter__(self):
    self._current = self
    return self

def __next__(self):
    if self._current is None:
        raise StopIteration
    else:
        val = self._current_value
        self._current = self._current.rest
        return val
```

a. Explain what the code in the `__iter__` method is doing:

_____

b. When does `__next__` output a `StopIteration` *exception*?
(*Hint: consult the previous question!)*

_____

How does out implementation of `__next__` know when to output a `StopIteration` exception?

_____

c. What does `__next__` do if it *does not* output a `StopIteration` exception?

_____

FYI: An ***exception*** is an event which occurs during the execution of a program that disrupts the normal flow of the program's commands. An exception is a Python object that represents an error, but supports some special handling of that error.

3. Examine the following interaction in interactive Python:

```
>>> char_lst = LinkedList('a', LinkedList('b', LinkedList('c')))
>>> for item in char_lst:
...     print(item)
a
b
c
```

a. How does the for..each loop above differ from the one we saw earlier in this activity?

_____

b. What *special method* might the for..loop call each time through the loop?
(*Hint: consult the code from question 1).*

_____

c. When might the `__iter__` method be called? Circle one:

Beginning of the for..loop | After the for..loop ends
Never | Every iteration of the loop

d. Why might we no longer see a `StopIteration` exception in this code?

_____

Application Questions: Use the Python Interpreter to check your work

TBD.