CS417 Lab #13

Getting Started

Begin the lab by downloading these starting files:

- `linked_list.py`
- `enhanced_list.py`

Background

You are provided with an implementation of a simple linked list, with head **and** tail pointers. It provides the following methods:

- `__init__(data)` builds a list from the given `data`, which could be another `Linked_List` or an ordinary python list. If `data` is omitted, builds an empty list.
- `__str__()` returns a human-friendly string describing the list's contents.
- `__repr__()` returns a detailed programmer-oriented string describing the list's nodes.
- `first()` returns the value in the head node, or `None` if the list is empty.
- `last()` returns the value in the tail node, or `None` if the list is empty.
- `add_head(x)` adds `x` to the front of the list.
- `add_tail(x)` adds `x` to the end of the list.
- `size(x)` walks down the list, and counts the number of nodes. Returns the count.
- `find_node(x)` returns the `List_Node` that contains the value `x`, or returns `None` if the list doesn't contain it.

Your Tasks

0. VERY CAREFULLY study the code in `linked_list.py`. Familiarize yourself with the techniques used to walk along the list:

   Carefully look at the `size()` and `find_node()` methods.

---

1. Open `enhanced_list.py`, and implement the `sum()` method.

   This method walks down the list, and adds up the `_value` field in each node. It returns the total.

---

2. Implement the `average()` method. You don't have to walk down the list. Just divide the sum (i.e. `self.sum()`) by the number of nodes (i.e. `self.size()`).

---

3. Implement the `reversed()` method. Here, you should first create a result linked list:

```
result = Linked_List()
```

   Then, walk down the list. For each value, add it to the *front* of the result:

```
result.add_head(some value goes here)
```

   Finally, `return result`.

   Thought exercise: what if we added each value to the *tail* of the result instead?

---

4. Implement the `index_of(value)` method. Walk down the list, just like in the `size()` method. At each step, increment a counter, and check if `current._value` matches `value`. If it does, return the counter.

   If you visit all the nodes, and never find `value`, then return `-1`.

   Thought exercise: will this work correctly if the list is empty?

---

5. Implement the `at_index(index)` method.

   First, check whether `index` is negative, or is `>= self.size()`. If so, `return None` (unlike python lists, negative indexes are *not* allowed here).

   Otherwise, walk down the list, incrementing a counter. When the counter matches `index`, return the current node's value.

Turning in your work

To submit your work, go to `mycourses.unh.edu`, find cs417, find the lab, and upload `enhanced_list.py`. Submit whatever you have completed, at the end of the lab session. You can submit again until midnight, with no lateness penalty.