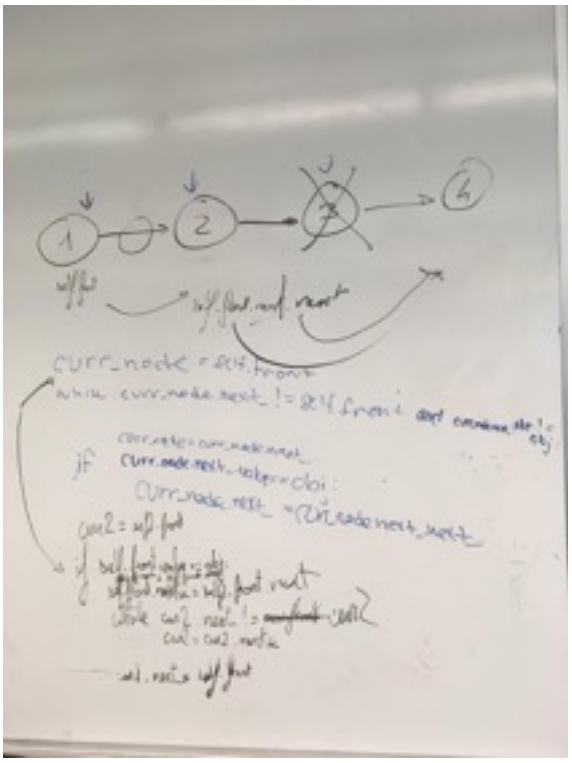
PEER INSTRUCTOR - REPORT

Mentor - Jesse Cordeiro
Mentee - Francois Gouelo

Date: June 22nd 2016: 5:45pm - 7:50pm, BA2210

First off, thank you Professor for organizing this, and thank you Jesse for being available and truly helpful.

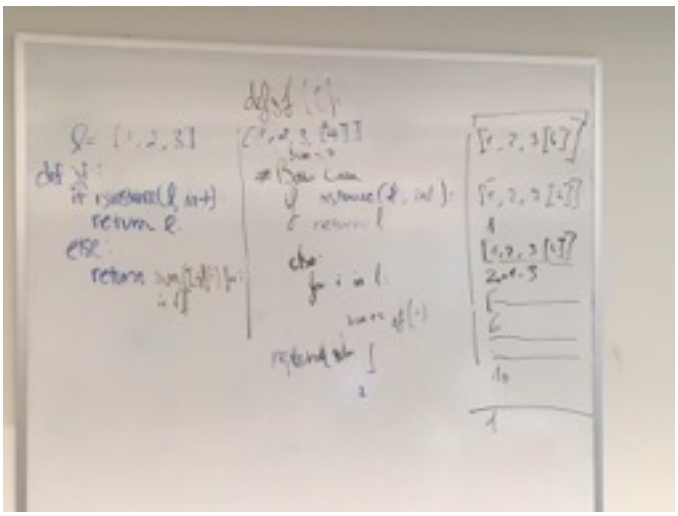
We started off our Peer Instruction session with Linkedlists. We did an overview of what and how it works. Then we looked at Circular Linkedlist. I had started writing code, but I was a bit stuck with the remove method.
We thought of all the different cases and exceptions. Jesse was very helpful as he would not give me solutions when I would not find something, but rather let me dig deeper to find the solution myself.



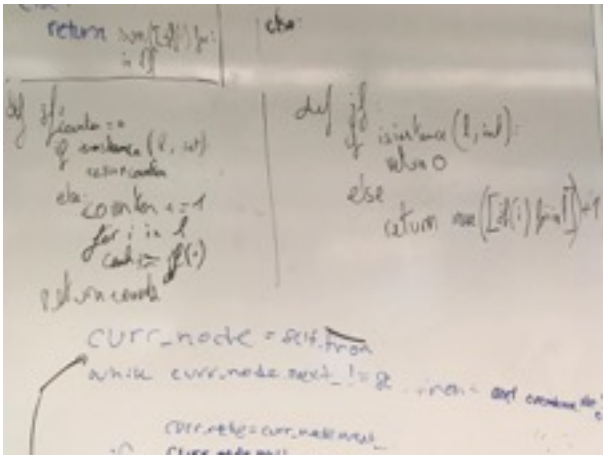
We went through a lot of ink and whiteboard space but eventually designed a proper code. What was very helpful was to write one “case”, and then test it out together to see if it worked. This took a bit of time to do, but was helpful in understanding what the next_ value represents, what creating curr_node = self.front does, and how to run through a linkedlist!

Now I have a deeper understanding of how to run through a linkedlist. I thought it was always just while curr_node.next is not None. But in this situation we have a circular linkedlist, so there would never be a None value, unless the linkedlist is None.
That was an interesting case to figure out. Furthermore, it forced be to think outside the box, and not focus so much on size and back, that I should not use in this class.

We then proceeded to look at a bit of recursion. We started it off with the sum of all elements in a list. The idea of working with recursion on a simple tasks was to look at list comprehension.
As you will notice in the picture below, we did two examples. The first one is if L is a list of integers (we are missing a picture of the non list comprehension implementation of this). The second examples is when we have a list in a list. I did the implementation rather quickly, but I was tracing it wrongly. I thought the summing was being done in the base case, when in fact it was done in the else case. So that was a good way for me to understand the true difference between base case and the recursive section of the function. Setting it in a list comprehension was me using my memory. That was not so productive, however it is the next example that really helped me start to understand how to put things in a list comprehension.



In this next examples we looked at the depth of elements. The depth of [1,2,3] is 1. The depth of [1,2,[3]] is 2, the depth of [1,2,[[[3]]]] is 4… <- it is the deepest depth.
In order to do so we used a similar process as the previous recursion. What truly changed everything was making it into a list comprehension. My understanding of list comprehension was limited to the function sum, so instead I was at first summing all the depths of each elements. After playing around, I understood I needed to add one somewhere. And after one or two very subtle hints, I figured out that the max function was the one I had to use.
This was really helpful, but I still need to practice on list comprehensions. I have a deeper understanding of it, but it remains fuzzy when I am faced with a complete new problem. I think this exercise cleared the fuzziness a bit, but once again: practice makes perfect!



Once again, thank you so much to Jesse and Professor Chinaei,
I will be doing another session as soon as possible,
Yours Sincerely,
Francois