206. Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list

① ② ③ ④

① → ② - ③ - ④ - ⑤ → ⑤ - ④ - ③ - ② - ①

Constraints: • The number of nodes in the list is the range [0, 5000] • -5000 <= Node.val <= 5000

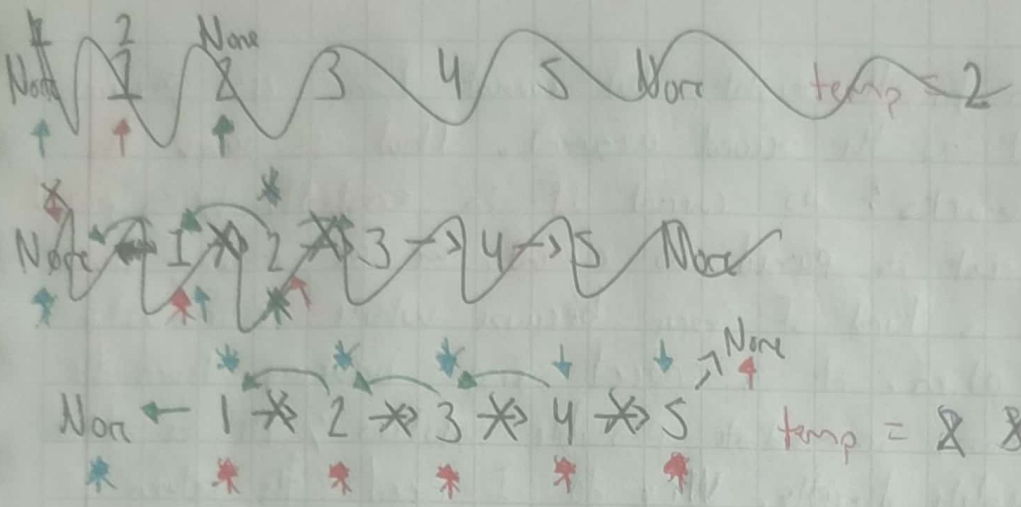
Definition: self.val = val ; self.next = next

My first instinct is to do some kind of bubble sort, in which I swap until everything is reversed. It would take n-1 loops, which honestly it isn't bad, the swaps would also take n-1. My problem here is that I find it extremely difficult to understand how can I tell my loop to end, as the ListNode doesn't has a method for length nor is possible to access its values with an index, such as an array. NeetCode talks about reversing the "links", but as my pointers are not explicit in Python (nor I probably want them to be) I can't change them directly. What I was thinking of doing was to go to the last node, and work recursively from it making its "next" node, actually the one that came before. I could write a function that goes all the way until my node is None, from there I could store the value to which I called the function as my next pointer. That would make a copy of the linked list, which isn't a problem for us now, as I want to solve the problem before bothering with efficiency.

	1	2	3	4	5
①	1	2	3 4	4	None

Current ~~Left~~ Previous ~~Left~~ temp

Current Previous ~~Right~~



None 1 2 3 4 5

To work with recursiveness I just need a function. In it I need two parameters, what ~~was~~ is the current node and which was the previous one (why could I have just one node? As that has access to both previous and next)

At the moment in which my current is None, I know I have reached the end of my list, and as I am reversing I know the previous pointer is the last one, so it can be set as the NEW HEAD. In case my current is not None yet, I need to make a swap: First, I take a reference to the pointer that is being used to connect current to my next (note that this is a singly linked list, so information just goes one way; instead of thinking it like a boridor I can thought of it as a highway, in which I can only have one direction). Second, I update that reference to point from current to previous (Not changing the direction of the highway but effectively rotating it 180° from the origin of current). I then repeat the whole process again, but now with the updated pointers: This is interesting, because I don't update the name of the variables themselves, but I do when passing the values as arguments. Next indeed

moves one position to the right, but current turns as previous when I passed it as the second argument. That is why the recursive function works: as current it is constantly being passed as previous and out is passed as current. We are moving them in a loop. It is kind of insane because while recursivity feels extremely alien, it is indeed just breaking down the problem to the most immediate possible solution; the one that is actually manageable directly. Why do I need the return statement? I suppose because the changes I have been doing have not "happened" without it... but how? If in my current is Node 1 I explicitly tell the function to return the new head, I never go out of the function, why the changes are not stored?