Programming and Data Structures with Python 2025

Assignment 3

17 Oct 2025, due 24 Oct 2025

User defined lists

The file `List.py` contains the user defined `List` class that we discussed in Lectures 15 and 16.

Modify this class to add the following functionality:

1. Write a function `member()` such that `l.member(v)` returns `True` if the value `v` is present in list `l` and `False` otherwise.
2. Write a function `valueat()` such that `l.valueat(i)` returns the value at index `i`. This is equivalent to `l[i]` for normal Python lists. As usual, valid indices run from `0` to `length(l) - 1`, and backwards from `-1` to `-length(l)`.

   The function should raise an `IndexError` if the index is not valid. The error message should include the index requested.
3. Modify the function `insert()` so that, for a list `l`, `l.insert(v,i)` inserts `v` before index `i`. In other words, after the insert, `l.valueat(i)` should be `v`. The index `i` is valid if it lies in the range `0` to `length(l)`. When `i` is `length(l)`, the effect is the same as `l.append(v)`.

   The default value for the index `i` is `0`. So, `l.insert(v)` should behave like `l.insert(v,0)`.

   The function should raise an `IndexError` if the index is not valid. The error message should include the index requested.
4. Write a function `deletelast()` such that `l.deletelast(v)` deletes the last occurrence of the value `v` from the list `l`.

   The function should raise a `ValueError` if the value to be deleted is not present in the list. The error message should include the value to be deleted.
5. Write a function `slice()` such that `l.slice(i,j)` returns a new list containing the slice from `l.valueat(i)` to `l.valueat(j-1)`.

   This is equivalent to `l[i:j]` for normal Python lists. The function `slice()` always takes two arguments. However, follow the usual list slicing conventions in case `i < 0`, `j <= i` or `j >= length(l)`. Do not generate any errors or raise any exceptions.
6. Write a function `rotate()` such that `l.rotate(k)` rotates the list `k` times in place — that is, without creating any new nodes.

   For instance if the list `l` holds the sequence `[1,2,3,4,5]`, `l.rotate(1)` would modify the sequence to `[5,1,2,3,4]`. After this update, `l.rotate(2)` would modify the sequence to `[3,4,5,1,2]`. Calling `l.rotate(2)` one more time would then restore the original sequence.

Instructions

- Submit your solution through Moodle as a Python notebook.
- Augment the given `List` class to include all the functionality asked above. All functions should work directly with the representation as a linked sequence of nodes. Do not, for example, export the user defined list into a Python list, manipulate the Python list and then repopulate the user defined list from the Python list.
- Add documentation to explain at a high level what your code is doing. If you have not implemented any of the functions above, mention this.
- Show some sample executions.