CSC148-Section:L0301/L0401 Week#4-Friday

Instructed by

AbdulAziz Al-Helali

a.alhelali@mail.utoronto.ca

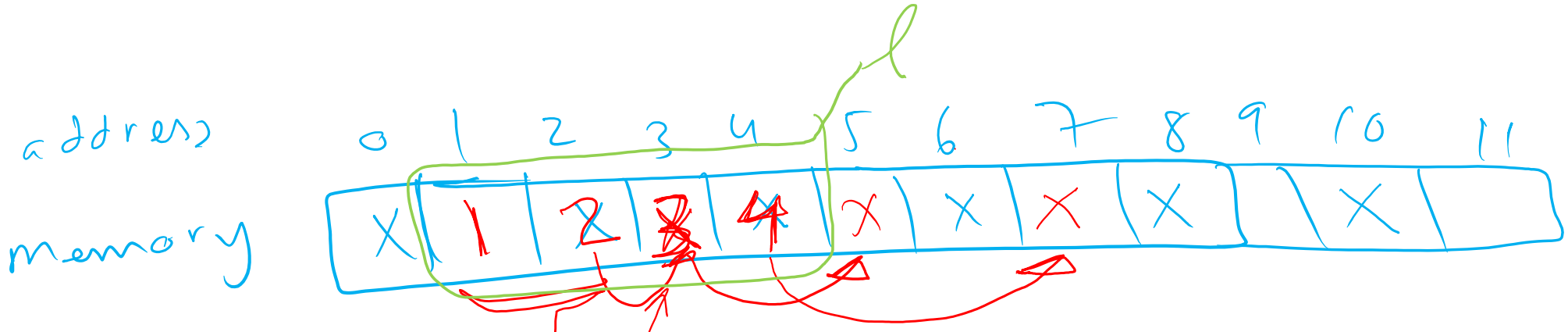
Office hours: Wednesday 11-1, BA2230.

Slides adapted from Professor Danny Heap course material winter17

Computer Science
UNIVERSITY OF TORONTO

Outline

- Linked Lists

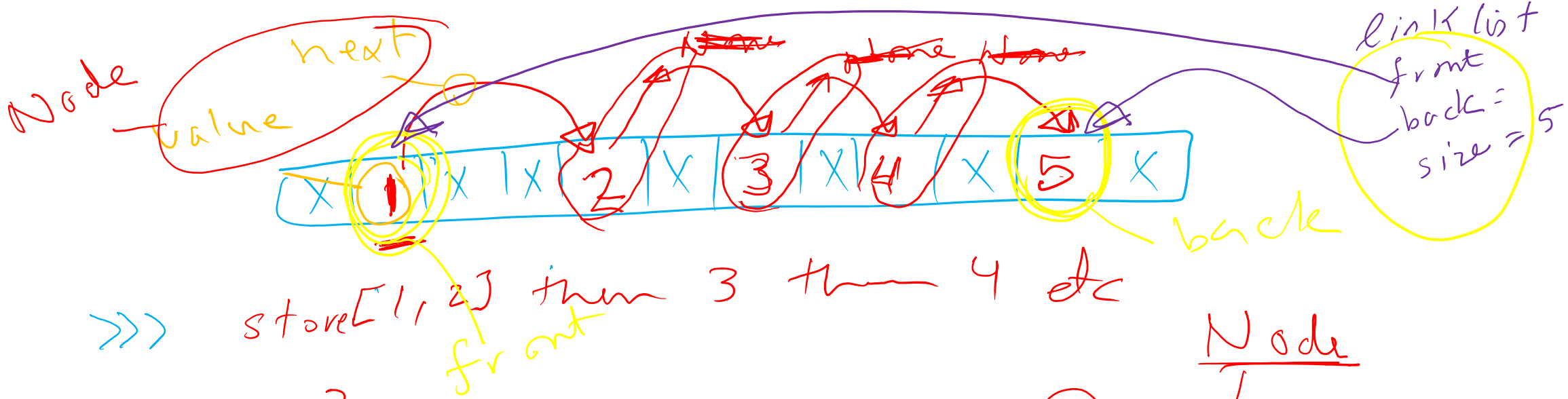



```
>>> l = [1, 2]
>>> l.append(3)
>>> l.append(4)
```

ask OS find adjacent memory plus

causes delay

Solutions? → linked list



>>> store[1, 2] then 3 then 4 etc

>>> 3

4

5

Store information about this list

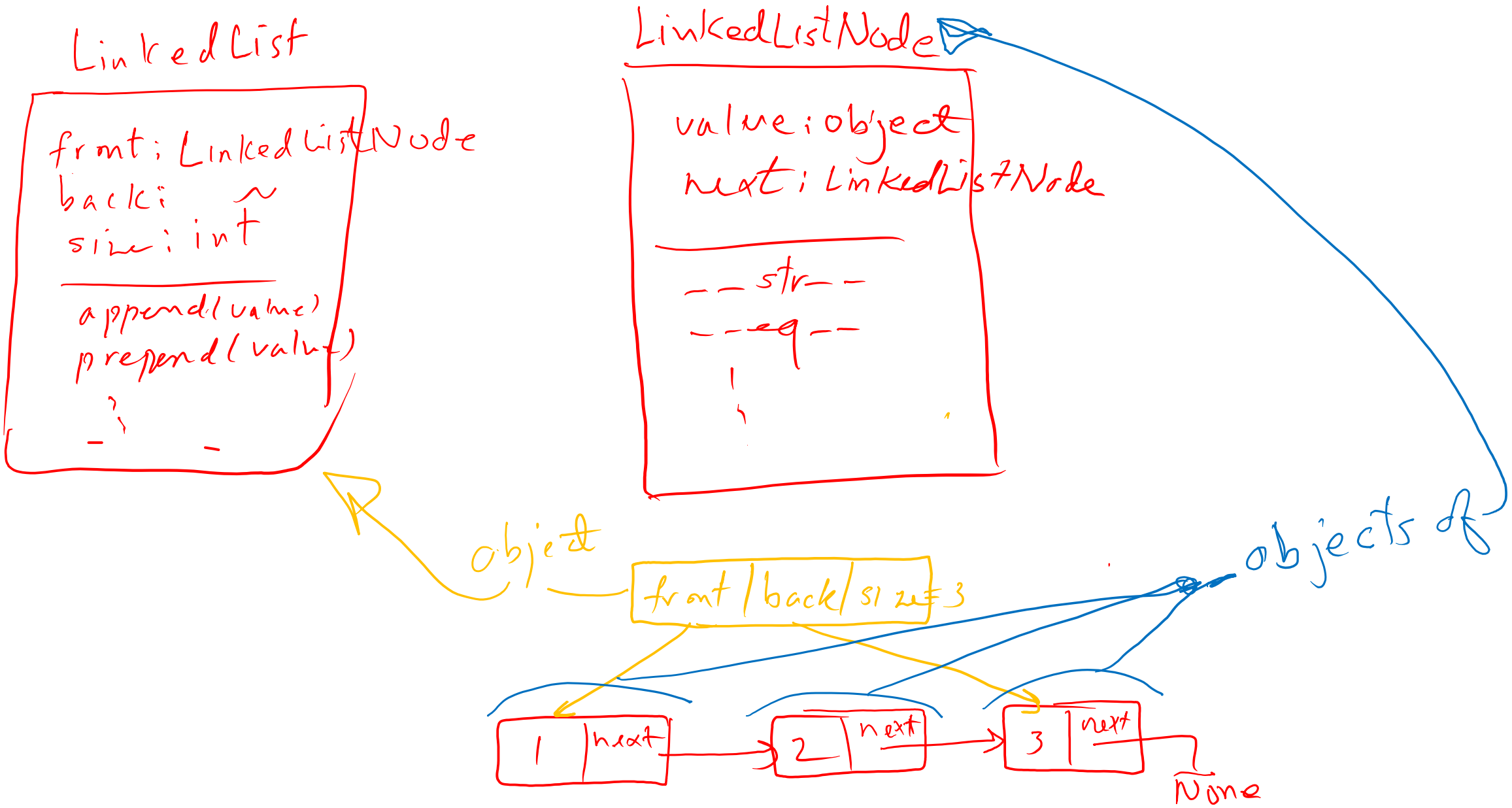
name
size
front
back

Node

value: object
next: Node

LinkedList

front: Node
back: Node
size: int

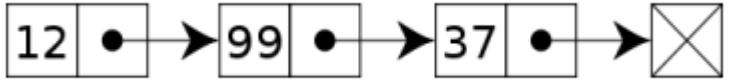

LinkedList
front: LinkedListNode
back:
size: int
append(value)
prepend(value)
LinkedListNode
value: object
next: LinkedListNode
__str__
__eq__
object
front | back | size = 3
objects of
1 | next
2 | next
3 | next
None

why linked lists?

- regular Python lists are flexible and useful, but overkill in some situations
- they allocate large blocks of **contiguous memory**, which becomes increasingly difficult as memory is in use.
- linked list nodes reserve just enough memory for the object value they want to refer to, a reference to it, and a reference to the next node in the list.

linked lists, two concepts

- There are two useful, but different, ways of thinking of linked list nodes
  1. as lists made up of an item (value) and a sub-list (rest)
  2. as objects (nodes) with a value and a reference to other similar objects



For now, will take the second point-of-view, and design a separate "wrapper" to represent a linked list as a whole.

a node class

```
class LinkedListNode:
    """
    Node to be used in linked list

    === Attributes ===
    value - data this LinkedListNode represents
    next_ - successor to this LinkedListNode
    """
    value: object
    next_: 'LinkedListNode'

    def __init__(self, value: object, next_:'LinkedListNode'=None)->None:
        """
        Create LinkedListNode self with data value and successor next_.
        """
        self.value, self.next_ = value, next_
```

a wrapper class for list

The list class keeps track of information about the entire list - such as its front, back, and size.

```
class LinkedList:
    """
    Collection of LinkedListNodes
    === Attributes ==
    front - first node of this LinkedList
    back - last node of this LinkedList
    size - number of nodes in this LinkedList a non-negative integer
    """
    front: LinkedListNode
    back: LinkedListNode
    size: int
    def __init__(self) -> None:
        """
        Create an empty linked list.
        """
        self.front, self.back, self.size = None, None, 0
```

Computer Science
UNIVERSITY OF TORONTO

division of labour

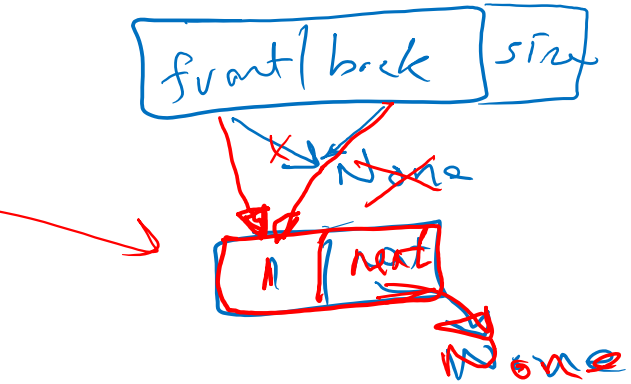
- Some of the work of special methods is done by the nodes:
- __str__
- __eq__
- Once these are done for nodes, it's easy to do them for the entire list.

prepend(value: object) → None 2 cases add to

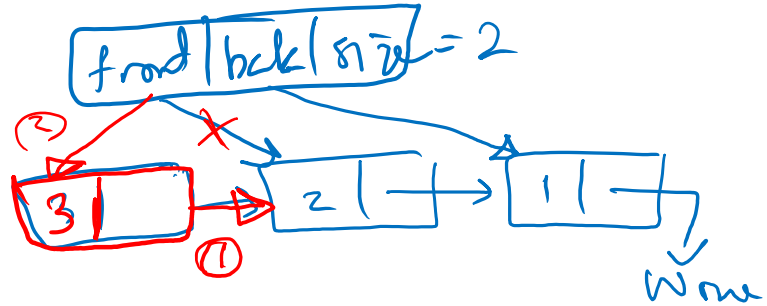
① Empty List

- create a new node and add it before self.front...

```
if size == 0
    n1 = LinkedListNode(value, None)
    front = n1
    back = n1
    size += 1
else
    n2 = LinkedListNode(value, front)   ①
    ② front = n2
    size += 1
```



② None Empty list



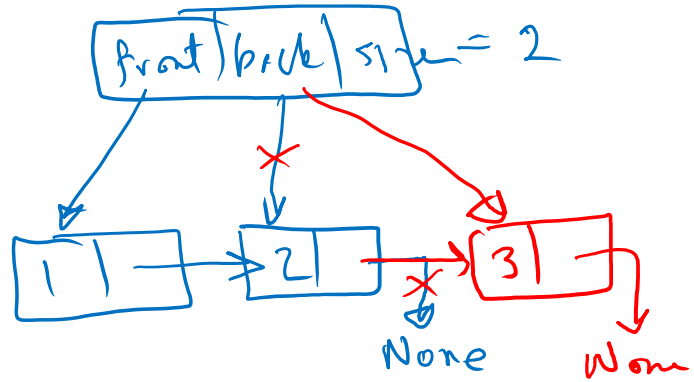
append (value: object) → None

2 cases

① like prepend.
see prev slide

② none empty list

```
if size == 0:
    n1 = LinkedListNode(value, None)
    front = n1
    back = n1
    size += 1
else:
    n2 = LinkedListNode(value, None)
    back.next = n2
    back = n2
    size += 1
```



Where Can I find the code presented in class

- You can find the full code in the course website under section **MWF2 (L0301) and MWF3 (L0401)**
- with the following file names:
  - Linked_list_Friday.py
  - (Note: the code has **no docstrings** and **might not be efficient** and it can be written in much better way. However, it is made this way with repetition of some lines to **keep you focused** on the concepts of **linked lists**)
- Download them Try different things with them and practice
  - Do not be afraid of doing mistakes

walking a list

- Make a reference to (at least one) node, and move it along the list:

```
cur_node = self.front
while <some condition here...>:
    # do something here...
    cur_node = cur_node.nxt
```