Outline

• Linked Lists
address

memory

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

Ask OS find adjacent memory plus causes delay

Solutions? → linked list
>>> store 1, 2, 3, 4 etc

3
4
5

Store information
about this list

name: size:
front: back:

Linked List
front: Node
back: Node
size: int

Node
value: object
next: Node
LinkedList

front: LinkedListNode
back: ~
siz: int

append(value)
prepend(value)

--- str ---
--- eq ---

LinkedListNode
value: object
next: LinkedListNode

object
front/back/siz: 3

objects of
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
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 \((\text{value}: \text{object}) \rightarrow \text{None}\)

2 cases add to

1) Empty List

\[\text{front} = \text{back} = \text{None}\]

2) None Empty List

\[\text{front} = \text{back} = \text{None}\]

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

1. If \(\text{size} = 0\)
   \[\text{n1} = \text{LinkedListNode}(\text{value}, \text{None})\]
   \[\text{front} = \text{n1}\]
   \[\text{back} = \text{n1}\]
   \[\text{size} = 1\]

2. Else
   \[\text{n2} = \text{LinkedListNode}(\text{value}, \text{front})\]
   \[\text{front} = \text{n2}\]
   \[\text{size} = 1\]
append

\[
(\text{value: object}) \rightarrow \text{None}
\]

2 cases

1. like prepend.
   see prev slide

2. none empty list

\[
\text{if size} = 0:
\]
\[
\begin{align*}
\text{\texttt{nil}} &= \text{LinkedListNode} (\text{value, None}) \\
\text{front} &= \text{\texttt{nil}} \\
\text{back} &= \text{\texttt{nil}} \\
\text{size} + &= 1
\end{align*}
\]

else:

\[
\begin{align*}
\text{\texttt{nil}} &= \text{LinkedListNode} (\text{value, None}) \\
\text{\texttt{back}.next} &= \text{\texttt{nil}} \\
\text{\texttt{back}.prev} &= \text{\texttt{nil}} \\
\text{size} + &= 1
\end{align*}
\]
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:

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