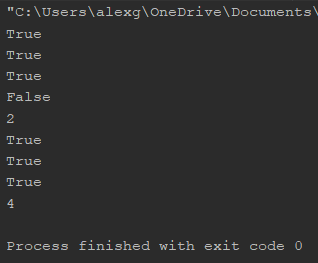
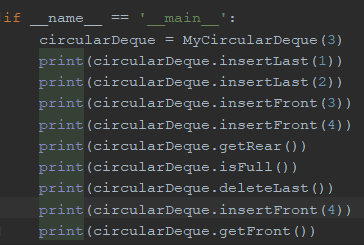
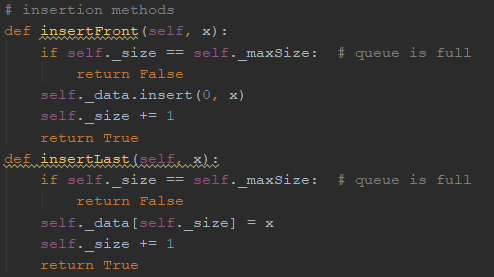
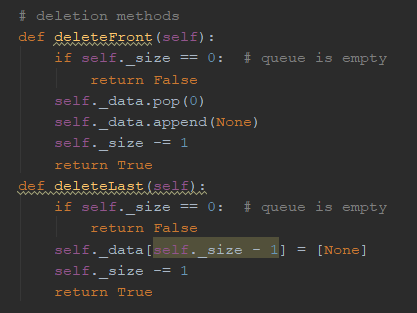
**Program 1**



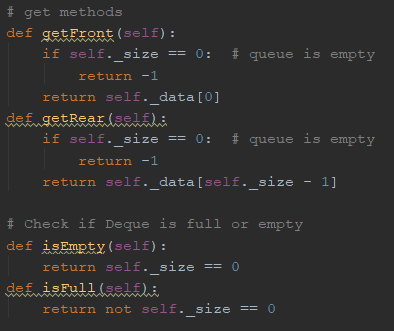
|  |  |
| --- | --- |
| **Class** | MyCircularDeque(k) |
| **Fields** | \_data, \_size, \_maxSize, \_front, \_back |
| **Behaviors** | insertFront(x), insertLast(x), deleteFront(), deleteLast(), getFront(), getRear(), isFull(), isEmpty() |



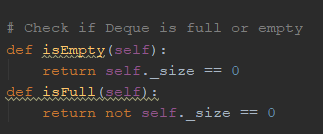
Both insertion methods first check if the queue is full. If room still exists, then x is added to the data. insertFront() uses the insert() method to move existing elements back 1 space and set index 0 to x. insertRear() checks the current size of the data and sets the index at the end of the data to x.



Both deletion methods first check if the queue is empty. If data still exists, then the deletion proceeds. deleteFront() pops the element at index 0 and then appends None to the end of the data (ensuring that the size remains constant). deleteLast() sets the index at the end of the data to None.

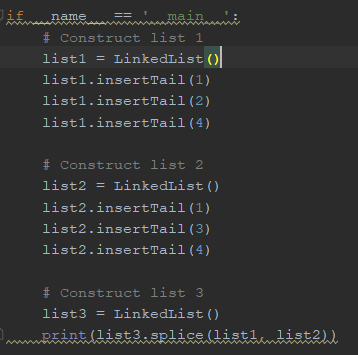


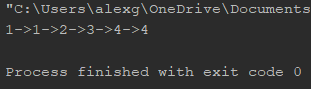
Both get methods first check if the queue is empty. If data still exists, then the element at the corresponding front or rear index is returned.



isEmpty() returns the result of the equivalence check between the data size and 0. isFull returns the opposite of this check.

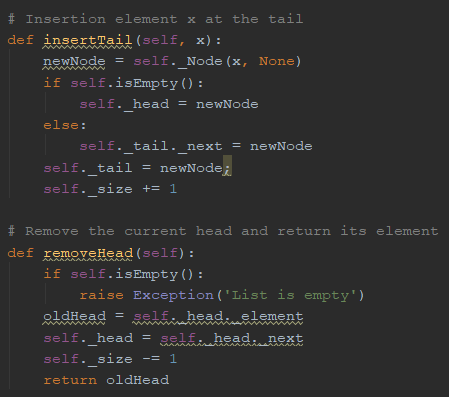
**Program 2**





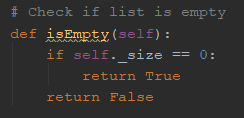
|  |  |
| --- | --- |
| **Class** | LinkedList() |
| **Fields** | \_head, \_tail, \_size |
| **Behaviors** | insertTail(x), removeHead(), isEmpty(), splice(l1, l2) |

|  |  |
| --- | --- |
| **Class** | \_Node |
| **Fields** | \_element, \_next |

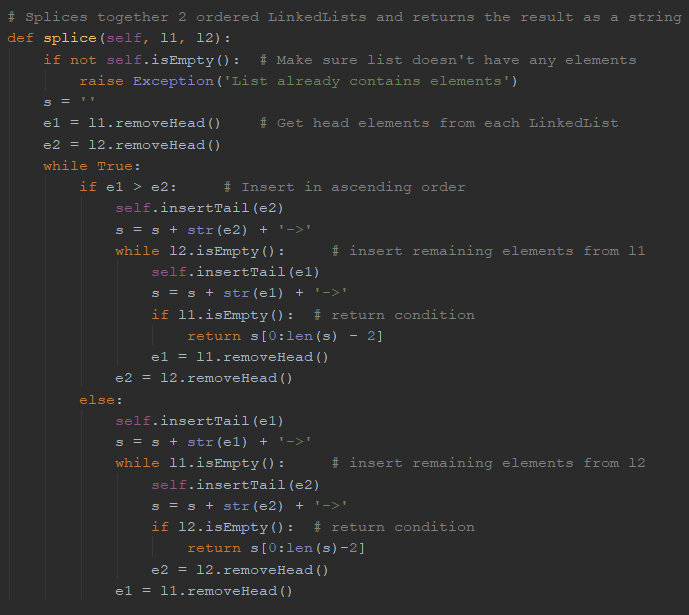


insertTail() creates a new node with x as the element. This new node is set as the tail, unless the list is empty, in which case it becomes the head.

removeHead() first checks if the list is empty, raising an exception if it is. Then, the node that the current head points to becomes the new head, and the old head is returned.

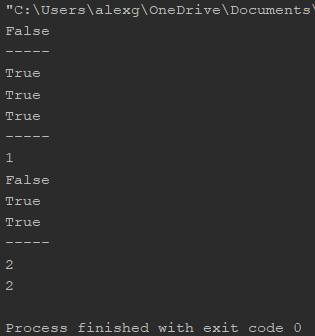
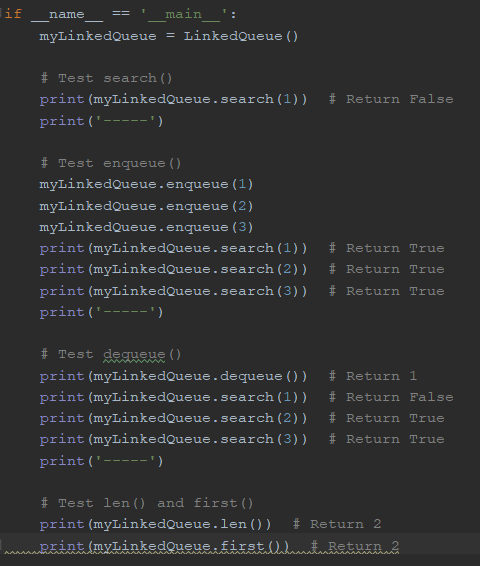


isEmpty() checks whether the current size is 0.



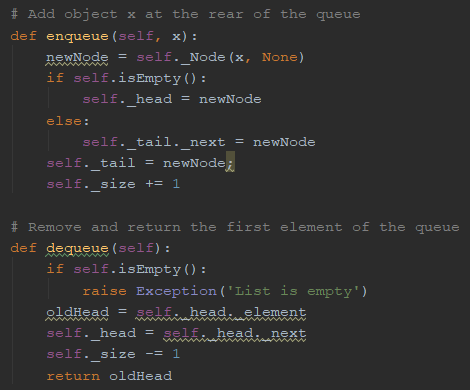
splice() first makes sure that the list is empty, raising an exception otherwise. The 2 heads of l1 and l2 are removed. Then, a loop is run. For each iteration of the loop, the lesser of the 2 heads is inserted into the spliced list, and the next head is removed. If 1 of the 2 lists is empty, the remaining elements from the other list are inserted into the new list. Once both lists are empty, a string containing the elements of the spliced list is returned, ending the loop.

**Program 3**



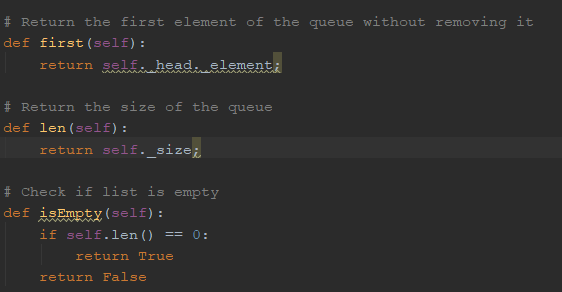
|  |  |
| --- | --- |
| **Class** | LinkedQueue() |
| **Fields** | \_head, \_tail, \_size |
| **Behaviors** | enqueue(x), dequeue(), first(), len(), isEmpty(), search(x) |

|  |  |
| --- | --- |
| **Class** | \_Node |
| **Fields** | \_element, \_next |



enqueue() creates a new node with x as the element. If the queue is empty, this new node becomes the head. Otherwise, the new node becomes the tail, with the previous tail pointing to it.

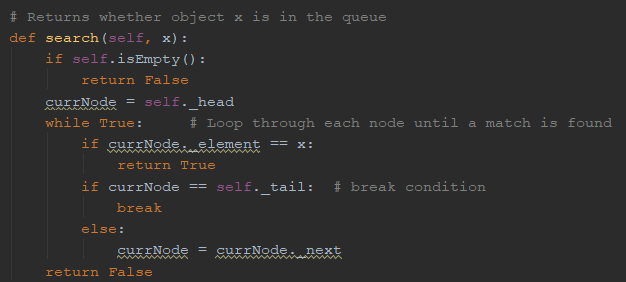
dequeue() first checks if the queue is empty, raising an exception if it is. Then, the node that the current head points to becomes the new head, and the old head is returned.



first() returns the element of the current head node.

len() returns the current size of the queue.

isEmpty() checks whether the current size is 0.



search() will immediately return False if the queue is empty, as x will certainly not exist in this case. Then, starting from the head, each node is checked for an element equal to x. If a match is found, search() returns True. If the loop completes and no match is found, search() returns False.