Assignment Uno

Jae Kyoung Lee (LJ)

Jae.Lee2@marist.edu

February 15, 2019

1 Stack

```
1  class stack(singleLinkedList):
2    def __init__(self, value):
3      super().__init__(value)
4
5    def isEmpty(self):
6      noValue = False
7      if self.head is None:
8        noValue = True
9      return noValue
10
11   def pop(self):
12     nodePointer = self.head
13     prevNodePointer = None
14     while nodePointer.next is not None:
15       prevNodePointer = nodePointer
16       nodePointer = nodePointer.next
17     print(nodePointer.value)
18     prevNodePointer.next = None
19
20   def push(self, data):
21     self.addValueEnd(data)
```

The function isEmpty validates whether the stack is empty or not by checking whether the head is equal to None. Line 12 creates a variable that assigns to head, which is used to locate the current position. Since unlike arrays, Linked List do not offer numeric indices, so this pointer is crucial to know where we are currently. Line 13 is another initialization of a variable, which is used to locate the previous Node and a temporary placeholder to hold the nodePointer. Line 14 is a while-loop that checks whether the current location is at the end or not. In lines 15-16: As we traversed, we have been assigning the prevNodePointer to the nodePointer and nodePointer to nodePointer.next. This means prevNodePointer holds the older value of the nodePointer and nodePointer now has a new value. Once we are at the end, in line 18, nodePointer should be pointing to a NULL value. However, the goal of this function is to remove the last element, so we assign a NULL value to prevNodePointer to indicate that prevNodePointer now is the last element. Line 21 adds a new value to the end of the stack.

2 Queue

```
1  class queue(singleLinkedList):
2    def __init__(self, value):
3      super().__init__(value)
4      self.queueHead = self.head
5
6    def isEmpty(self):
7      noValue = False
8      if self.head is None:
9        noValue = True
10     return noValue
11
12   def enqueue(self, data):
13     queueCounter = 0
14     while queueCounter < 100:
15       newEmtpyNode = node(None)
16       newEmtpyNode.next = self.head
17       queueCounter+=1
18     queueLength = queueCounter
19     nodePointer = self.head
20     lengthCounter = 0
21     while (lengthCounter < queueLength) and (nodePointer.next is not None):
22       nodePointer = nodePointer.next
23       lengthCounter+=1
24     nodePointer = self.queueHead
25     self.addValueEnd(data)
26
27   def dequeue(self):
28     nodePointer = self.head
29     nodePointer = nodePointer.next
```

Line 12 initializes a counter variable that will be used at the while-loop in line 14. the While-loop in line 14 sets a maximum and static queue size. After the while-loop ends, it sets the variable ' queueLength ' to the counter, which is used to check and make sure that the queue doesn't go over the set length.