Aufgabe 1

Deque steht für *Double Ended Queue*.

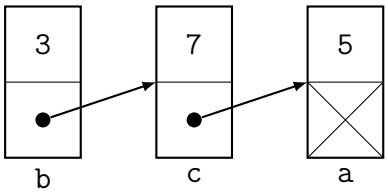
Aufgabe 2

- Hinzufügen eines Elements am Anfang der Deque
  `addFront(element)`
- Entfernen eines Elements am Anfang der Deque
  `removeFront()`
- Hinzufügen eines Elements am Ende der Deque
  `addRear(element)`
- Entfernen eines Elements am Ende der Deque
  `removeRear()`
- Test auf Leerheit
  `isEmpty()`
- Anzahl Elemente in der Deque
  `size()`

Aufgabe 3

```
1  from deque import Deque
2
3  def palinChecker(text):
4      d = Deque()
5      for zeichen in text:
6          d.addFront(zeichen)
7      print(d)
8      while d.size()>1:
9          if d.removeFront() != d.removeRear():
10             return False
11     return True
```

Aufgabe 4



Die Andordnung der drei Knoten in der Ebene ist unwichtig, so lange die Referenz eines Knotens auf den richtigen Zielknoten zeigt. Ebenso spielt es keine Rolle, ob innerhalb eines Knotens die Daten oben und die Referenzen unten oder umgekehrt positioniert sind.

Aufgabe 5

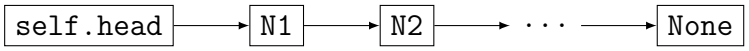
Der Code fügt zwar in Zeile 26 den in Zeile 25 erzeugten neuen Knoten an den Kopf der Liste an, „vergisst“ aber, den neuen Knoten mit der bisherigen Liste zu verbinden, indem der Wert von `self.head` in die Zeigervariable des neuen Knotens kopiert wird.

```
1     def add(self, item):
2         temp = Node(item)
3         temp.setNext(self.head)
4         self.head = temp
```

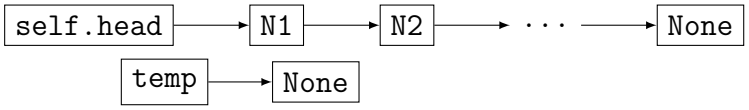
In Zeile 3 des obigen korrekten Codes wird die Adresse des alten ersten Knotens in die `next`-Variable des neuen Elements kopiert. Danach kann in der Variablen `self.head` die Adresse des neuen Knotens gespeichert werden.

Visualisierung:

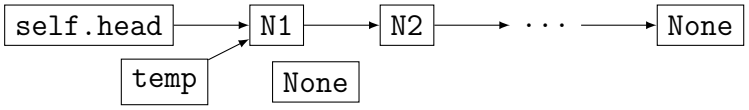
1. Liste vor dem Ausführen der `add`-Methode:



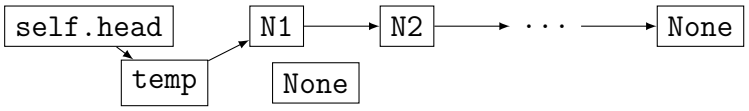
2. Nach dem Ausführen von Zeile 2:



3. Nach dem Ausführen von Zeile 3:



4. Nach dem Ausführen von Zeile 4:



Der verwaiste Wert `None` wird irgendwann vom Garbage Collector erkannt, der den Speicherplatz wieder freigibt.

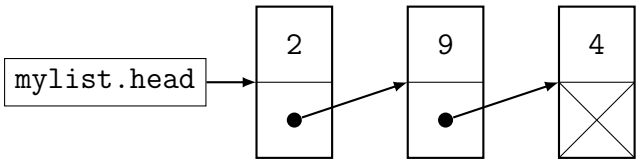
Aufgabe 6

(a)	Entfernen eines Elements an beliebiger Position	$O(n)$
(b)	Hinzufügen eines Elements am Anfang der Liste	$O(1)$
(c)	Durchsuchen der Liste nach einem Element	$O(n)$
(d)	Bestimmen der Länge der Liste	$O(n)$
(e)	Testen, ob die Liste leer ist	$O(1)$

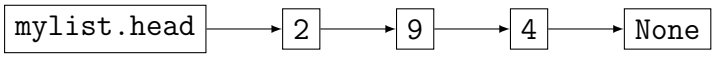
Aufgabe 7

Garbage Collection ist ein automatisiertes Verfahren, das die von einem Programm nicht mehr benutzten Speicherbereiche („Garbage“) erkennt und diese dem Programm wieder zur Verfügung stellt.

Aufgabe 8

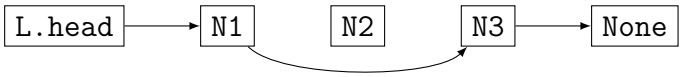


oder ohne detaillierte Darstellung der Zeigerreferenzen:



Aufgabe 9

1. Neben dem Knoten `N2` muss auch sein Vorgänger `N1` bestimmt werden.
2. Dann muss der Wert von `N2.next` der Variable `N1.next` zugewiessen werden.
3. Danach zeigt der Knoten `N1` auf den Knoten `N3`. Grafisch:



Der Garbage Collector findet früher oder später den verwaisten Knoten `N2` und gibt dessen Speicherplatz wieder frei.

Aufgabe 10

```
1     def search(self, item):
2         ref = self.head
3         while ref != None:
4             if ref.data == item:
5                 return True
6             else:
7                 ref = ref.getNext()
8         return False
```