Aufgabe 1

Beschreibe möglichst viele strukturellen Merkmale der Datenstruktur *Queue*?

Aufgabe 2

Beschreibe die typischen Operationen der Datenstruktur *Queue*.

Aufgabe 3

Gib zwei unterschiedliche Anwendungen der Datenstruktur *Queue* an.

Aufgabe 4

Warum eignen sich Python-Listen nur bedingt zur Implementierung von Queues?

Aufgabe 5

Welche Ausgaben macht der folgende Python-Code?

```
from queue import Queue
q = Queue()
q.enqueue(7)
q.enqueue(2)
q.enqueue(3)
q.enqueue(5)
x = q.dequeue()
q.enqueue(1)
q.enqueue(4)
y = q.dequeue()
print(x)
print(y)
print(q.size())
q.enqueue(q.dequeue())
q.enqueue(q.dequeue())
z = q.dequeue()
print(z)
```

Aufgabe 6

Welche Ausgabe macht der folgende Code, der Node-Objekte mit einem Zeiger auf Nachbarknoten verküpft?

```
class Node:

    def __init__(self, item, next = None):
        self.item = item
        self.next = next

    def get_item(self):
        return self.item

    def get_next(self):
        return self.next


c = Node(5)
a = Node(7)
d = Node(3)
e = Node(9)
b = Node(4)

e.next = a
d.next = b
c.next = e
b.next = c

print(d.get_next().get_next().get_next().get_item())
```

Aufgabe 7

Vervollständige das Zeigerdiagram der Queue `q` anhand des Speicherabbildes, das noch andere Daten enthält. Ein Knotenobjekt besteht jeweils aus zwei aufeinanderfolgenden Feldern, wobei im ersten Feld der Wert und im zweiten die Referenz auf das erste Feld des nächsten Knoten steht. `00` steht für `None`. Welche Daten befinden sich in welcher Reihenfolge in der Queue?

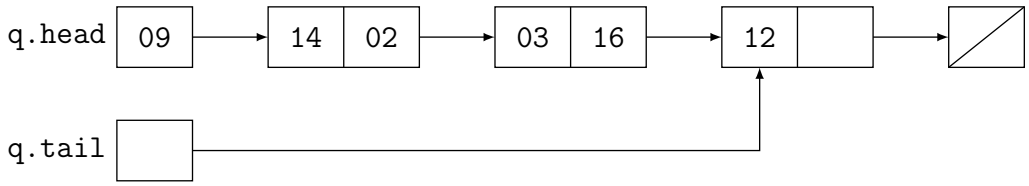
	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9
0.		06	00	08	16	15	00	15	00	09
1.	15	11	18	01	16	06	04	07	16	00

`q.head` ☐

`q.tail` ☐

Aufgabe 8

Gegeben ist folgendes Zeigerdiagramm einer Queue `q`.



(a) Trage die passenden Werte in die leeren Felder ein.

(b) Skizziere das Zeigerdiagramm nach einer `dequeue`-Operation auf `q`.

(c) Erstelle ein Speicherabbild der Queue `q`, nachdem (b) ausgeführt wurde. Berücksichtige auch den verwaisten Knoten.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9
0.										
1.										

Aufgabe 9

(a) Implementiere die Methoden `queue(item)` und `dequeue()` der Klassse `Queue` in Python auf der Basis einer Liste.

Hinweis: Es gibt mehrere korrekte Lösungen aber es genügt, eine anzugeben.

```
class Queue:

    def __init__(self):
        self.data = []
```

(b) Gib die Laufzeitkomplexität der von dir in Python implementierten Methoden an, wenn die aktuelle Queue n Elemente enthält.