Datenstrukturen: Queue (Warteschlange)

Das Konzept

Eine *Queue* oder *Warteschlange* ist eine Datenstruktur, in der die Elemente in der gleichen Reihenfolge entnommen werden, in der sie ihr hinzugefügt wurden. Dies wird durch die Formel *First In – First Out* prägnant ausgedrückt. Methoden:

- Eine leere Queue erzeugen: `q = Queue()`
- Ein Element hinzufügen: `q.enqueue(item)`
- Ein Element entnehmen: `q.dequeue()`
- Die aktuelle Grösse der Queue zurückgeben: `q.size()`
- Teste, ob die Queue leer ist: `q.empty()`
- Eine Queue leeren: `q.clear()`

Anwendungen

- *Puffer für Tastatur- und Mauseingaben:* Da Mensch und Computer unterschiedliche Verarbeitungsgeschwindigkeiten haben benötigt es einen Zwischenspeicher, der nach dem FIFO-Prinzip arbeitet.
- *Interprozesskommunikation:* Wenn die Ausgabe eines Programmes in die Eingabe eines anderen Programmes umgeleitet wird.
- *Druckerwarteschlangen:* Queues erfüllen das Bedürfnis nach Fairness. Wenn A vor B einen Druckauftrag an einen gemeinsam genutzten Drucker sendet, so sollte A den Ausdruck auch vor B erhalten.

Realisierung mit einer verketteten Liste

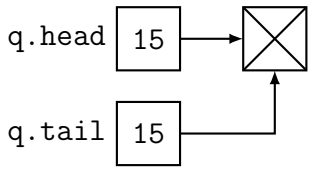
Da wir diese Datenstruktur am Ende des Dokuments mit Python implementieren und sich Python-Listen dafür nicht besonders gut eignen, verwenden wir ein Verfahren, bei der jedes Element der Warteschlange zusammen mit einer Referenz (Adresse) auf den nächsten solchen Container gespeichert wird.

Wir zeigen, wie eine leere Queue erzeugt, wie Elemente am Ende der Warteschlange hinzugefügt und danach von ihrem Kopf wieder entfernt werden. Diese Vorgänge werden durch Grafiken, mit den Daten und Speicheradressen illustriert. Damit man Adressen und Daten besser auseinander halten kann, stehen Zahlen für Adressen und Buchstaben für Daten. Ein Quadrat mit zwei Diagonalen bezeichnet den leeren Zeiger, der je nach Programmiersprache `None`, `NUL` oder `NIL` genannt wird.

Achtung: Die Pfeile in den Grafiken zeigen, wo Python das nächste Element findet. Die Elemente „wandern“ aber in entgegengesetzter Richtung durch die Datenstruktur

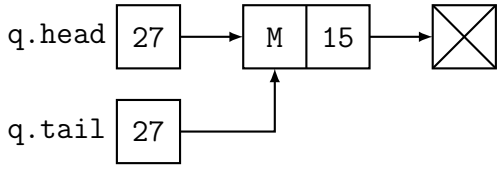
Erzeugen einer leeren Queue

`q = Queue()` Hinweis: Die Adressen der der Speicherzellen (15, 27, ...) wurden willkürlich gewählt.



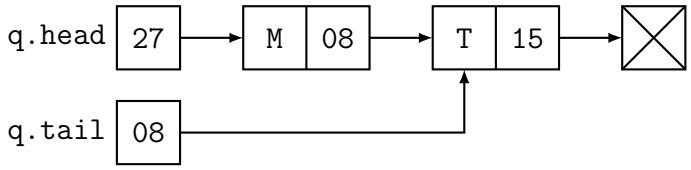
Einfügen eines ersten Elements

`q.enqueue(M)`



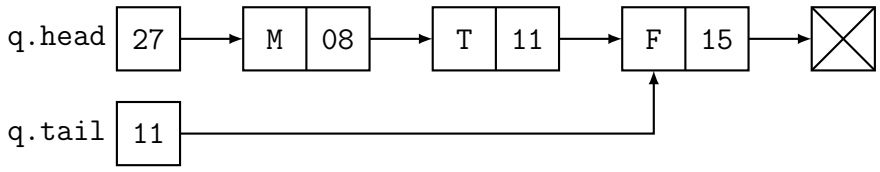
Einfügen eines zweiten Elements

`q.enqueue(T)`



Einfügen eines dritten Elements

`q.enqueue(F)`



Speicherabbild nach der letzten Operation

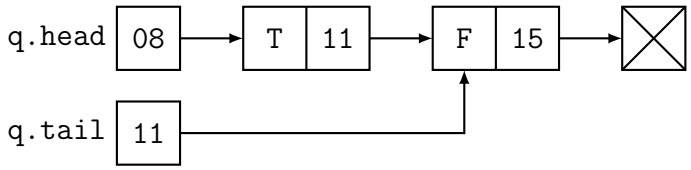
Datenknoten besteht jeweils aus einem Wert und einer Referenz auf den nächsten Datenknoten. Der Wert `00` stellt `None` dar. Ferner ist `q.head = 27` und `q.tail = 11`.

Damit man die Ziffern der Adresse nicht verwechselt, haben Spaltenindizes einen Punkt links und Zeilenindizes einen Punkt rechts.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9
0.									T	11
1.		F	15			00				
2.								M	08	

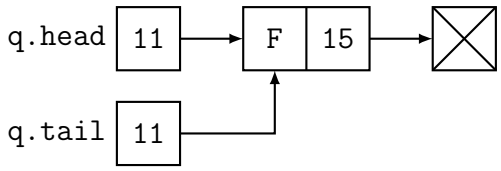
Entfernen des vordersten Elements

`q.dequeue()` ⇒ `M`



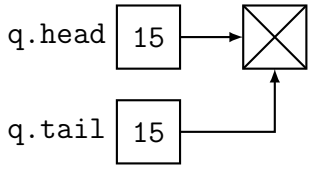
Entfernen des vordersten Elements

`q.dequeue()` ⇒ `T`



Entfernen des vordersten Elements

`q.dequeue()` ⇒ `F`



Python-Implementierung

```python
class Node:

    def __init__(self, obj, nxt=None):
        self.obj = obj
        self.nxt = None

class Queue:

    def __init__(self):
        '''Erzeugt eine neue leere Queue'''
        self.head = None
        self.tail = None
        self.size = 0 # "eager"

    def __len__(self):
        '''Gibt die Länge der Queue zurück'''
        return self.size

    def enqueue(self, obj):
        '''Fügt ein Objekt am Ende ein'''
        new_node = Node(obj)
        if self.head == None:
            self.head = new_node
            self.tail = new_node
        else:
            self.tail.nxt = new_node
            self.tail = new_node
        self.size += 1

    def dequeue(self):
        '''Entfernt Element am Anfang und gibt es zurück'''
        if self.head == None:
            return None
        obj = self.head.obj
        self.head = self.head.nxt
        self.size -= 1
        if self.head == None:
            self.tail = None
        return obj

    def is_empty(self):
        '''Gibt True zurück, wenn Queue leer ist; sonst False'''
        return self.head == None
```