Prof. Dr. Goran Glavaš,
M.Sc. Fabian David Schmidt
M.Sc. Benedikt Ebing
Lecture Chair XII for Natural Language Processing, Universität Würzburg

1. Exercise for "Algorithmen, KI & Data Science 1"

1 Queue

1. Name three real world applications of queue.

   - Car wash queue
   - Ski lift queue
   - Printer queue
   - Escalator

2. Illustrate the result of each operation in the sequence ENQUEUE(4), ENQUEUE(1), ENQUEUE(3), DEQUEUE(), ENQUEUE(8), DEQUEUE() on an initially empty queue.

```
4
4  1
4  1  3
   1  3
   1  3  8
      3  8
```

3. Implement your own list-based queue-class in Python. You'll find a template of the class in the corresponding .ipynb file provided along with the exercises sheet.

Implement the following methods:

a) *isEmpty*: The method should return true, if the queue is empty and false otherwise.

b) *enqueue*: The method should receive an item as input and append the item to the end of the queue.

c) *dequeue*: The method should remove the first element from the queue and return it to the caller. If the queue is empty, the method should return "Queue is empty".

d) *getHead*: The method should return the first element of the queue. If the queue is empty, the method should return "Queue is empty".

e) *getTail*: The method should return the last element of the queue. If the queue is empty, the method should return "Queue is empty".

f) *size*: The method should return the size of the queue.

Hint: You can check the Python documentation of *list* for additional help: `https://docs.python.org/3/tutorial/datastructures.html#more-on-lists`

2 Stack

1. Name three real world applications of stacks.

   - Stack of tablets in Mensa
   - Stack of Pringles^{TM}
   - Stack of visited websites in your browser

2. Illustrate the result of each operation in the sequence PUSH(4), PUSH(1), PUSH(3), POP(), PUSH(8), POP() on an initially empty stack.

```
4
4  1
4  1  3
4  1
4  1  8
4  1
```

3. Explain in your own words how to implement a stack using two queues.

   The following is a way of implementing a stack using two queues, where pop takes linear time, and push takes constant time. The first of these ways, consists of just enqueueing each element as you push it. Then, to do a pop, you dequeue each element from one of the queues and place it in the other, but stopping just before the last element. Then, return the single element left in the original queue.

4. Implement your own list-based stack-class in Python. You'll find a template of the class in the corresponding .ipynb file provided along with the exercises sheet. Implement the following methods:

   a) *isEmpty*: the method should return true, if the stack is empty and false otherwise.

   b) *push*: the method should receive an item as input and appends the item to the end of the stack.

   c) *pop*: the method should remove the last element from the stack and return it to the caller. If the stack is empty, the method should return "Stack is empty".

   d) *peek*: the method should return the last element of the stack. If the stack is empty, the method should return "Stack is empty".

   e) *size*: the method should return the size of the stack.

   Hint: You can check the Python documentation of *list* for additional help: `https://docs.python.org/3/tutorial/datastructures.html#more-on-lists`