Datenstrukturen (Kapitel 2) **Lösungen+** **Prüfungsvorbereitung**

Aufgabe 1

```
class Stack:

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

    def push(self, item):
        self.data.append(item)

    def pop(self):
        return self.data.pop()

    def peek(self):
        return self.data[-1]

    def size(self):
        return len(self.data)

    def isEmpty(self):
        return self.data == []
```

Aufgabe 2

```
3
7
```

Aufgabe 3

Für die Funktionsweise eines Stacks: *Last In – First Out*

Aufgabe 4

- Undo/Redo-Funktionen von Anwenderprogrammen (Browser-History)
- Auswertung von Ausdrücken (z. B. UPN) und Parsen von Syntax (z. B. Ausdrücke auf korrekte Verschachtelung prüfen)
- Verwaltung des Arbeitsspeichers von Computern
- Als Datenstruktur für Algorithmen (z. B. Lösungsalgorithmen für Irrgärten)

Aufgabe 5

(a) `4 3 2 1 0 9 8 7 6 5`

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

`4 3 2 1 0 9 8 7 6 5` `True`

(b) `4 6 8 7 5 3 2 9 0 1`

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

`4 6 8 7 5 3 2 9 1 0` `False`

(c) `2 5 6 7 4 8 9 3 1 0`

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

`2 5 6 7 4 8 9 3 1 0` `True`

(d) `4 3 2 1 0 5 6 7 8 9`

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

`4 3 2 1 0 5 6 7 8 9` `True`

Aufgabe 6

(a) `0 1 2 3 4 5 6 7 8 9` True

(b) `4 6 8 7 5 3 2 9 0 1` False

(c) `2 5 6 7 4 8 9 3 1 0` False

(d) `4 3 2 1 0 5 6 7 8 9` False

Wenn die Elemente in aufsteigender Reihenfolge in die Queue eingereiht werden, dann müssen sie auch in der gleichen Reihenfolge wieder die Queue verlassen (First In – First Out). Daher ist für die Inputfolge `[0, 1, 2, ..., 8, 9]` nur (a) möglich.