Datenstrukturen: Bag (Multimenge) **Lösung+** **Prüfungsvorbereitung**

Aufgabe 1

Es handelt sich um eine Sammlung von Elementen, in der

- die Reihenfolge der Elemente unwesentlich ist,
- Elemente wiederholt vorkommen können.

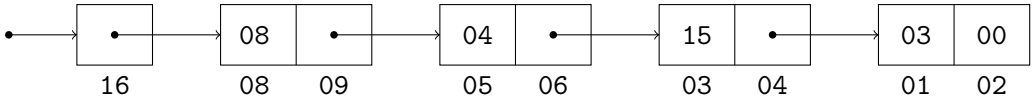
Aufgabe 2

- einen leeren Bag erzeugen
- ein Element zum Bag hinzufügen
- den Bag durchlaufen
- den Bag löschen
- die Anzahl der Elemente des Bags zurückgeben
- ein Element aus dem Bag entfernen (optional)

Aufgabe 3

- Primfaktorzerlegung ($72 = 2 \cdot 2 \cdot 2 \cdot 3 \cdot 3$)
- Datenbanken (Speicherung gleicher Datensätze)
- Statistik (Mittelwertberechnungen)

Aufgabe 4



Um den Wert `04` zu löschen, muss die in der Zelle `09` stehende Adresse `05` mit der Adresse `03` überschrieben werden.

Aufgabe 5

```
1  def gf(a, q, n):
2      for k in range(0, n):
3          yield a * q**k
4
5  for x in gf(3, 2, 4):
6      print(x)
```

Das Programm gibt 3, 6, 12, 24 aus. `yield` liefert bei jedem Aufruf einen Rückgabewert. Anders als bei `return` stoppt die Funktion nicht sondern wird bis zum Ende durchlaufen. So wird eine Folge von Werten generiert, über die Python in Zeile 5 iterieren kann.

Aufgabe 6

```
1  class Node:
2
3      def __init__(self, item, nxt):
4          self.item = item # Nutzdaten
5          self.nxt = nxt   # Zeiger
6
7  class Bag:
8
9      def __init__(self):
10         self.first = None # Zeiger auf 1. Knoten
11         self.n = 0        # Anzahl Elemente

13     def add(self, item):
14         # neuer Knoten mit Zeiger auf alten Anfang:
15         tmp = Node(item, self.first)
16         # neuer Anfang zeigt auf neuen Knoten:
17         self.first = tmp
18         # Grösse anpassen
19         self.n += 1
```

Aufgabe 7

- `Bag()` $O(1)$
- `add(item)` $O(1)$
- `__iter__()` $O(n)$
- `clear()` $O(1)$
- `size()` $O(1)$

Aufgabe ?

```
1 def af(a, d, n):
2     for k in range(0, n):
3         yield a + k*d
4
5 for x in af(17, -3, 5):
6     print(x)
```

Es werden die ersten 5 Glieder einer arithmetischen Folge ausgegeben:

17, 14, 11, 8, 5

`yield` liefert bei jedem Aufruf einen Rückgabewert. Anders als bei `return` stoppt die Funktion nicht sondern wird bis zum Ende durchlaufen. So wird eine Folge von Werten generiert, über die Python in Zeile 5 iterieren kann.

Aufgabe ?

(a)
```
def clear(self):
    self.first = None
    self.n = 0
```

(b)
```
def add(self, item):
    self.first = Node(item, self.first)
    self.n += 1
```

(c) 5, 9, 3, 7 in dieser Reihenfolge