1. Du kannst angeben, wie viele Vergleiche im Mittel nötig sind, um einen Wert in einer (unsortierten) Liste der Länge n zu finden.

2. Du kannst mit Hilfe der Modulo-Funktion aus einem numerischen Schlüssel den zugehörigen Hashwert berechnen. In den Aufgaben sind die Zahlen so gewählt, dass diese Rechnungen ohne Taschenrechner durchgeführt werden können.

3. Du kannst Hashwert-Kollisionen mit linearem Sondieren (linear probing) auflösen.

4. Du kannst Hashwert-Kollisionen mit Verkettung (chaining) auflösen.

5. Du kannst jeweils einen Vor- und einen Nachteil des linearen Sondierens und des Chainings beschreiben.

6. Du kannst den Auslastungsgrad α (*load factor*) für Hashtabellen mit linearem Sondieren und und mit Verkettung bestimmen:

   - lineares Sondieren: $\alpha = \dfrac{\text{Anzahl verwendeter Schlüssel}}{\text{Anzahl möglicher Behälter}}$
   - Chaining: $\alpha = \dfrac{\text{Anzahl verwendeter Schlüssel}}{\text{Anzahl der Listen}}$

   Achtung: unterschiedliche Wertebereiche

7. Du kannst im Falle des linearen Sondierens und des Chainings angeben, für welche Lastfaktoren ein Rehashing sinnvoll ist.

Hashtabelle ...	lineares Sondieren	Verketten
halbieren	$\alpha < \frac{1}{8}$	$\alpha < 2$
verdoppeln	$\alpha > \frac{1}{2}$	$\alpha > 8$

8. Du kannst die amortisierte[1] Laufzeitkomplexität für das Suchen eines Schlüssels in einer ausgeglichenen Hashtabelle angeben. [Sie ist $O(1)$.]

9. Du kannst beschreiben, wie Zeichenketten-Schlüssel möglichst eindeutig in ganze Zahlen transformiert werden können, damit Hashfunktionen auf sie angewendet werden können.

10. Du kannst den Namen der Python-Datenstruktur angeben, die Hashing bzw. Hashtabellen verwendet.

[1] Anstelle der Worst Case-Laufzeit wird die mittlere Worst Case-Laufzeit über mehrere Durchläufe bestimmt, was zu tieferen mittleren Laufzeiten führt, wenn der Worst Case nur sehr selten auftritt.