Datenstrukturen **Prüfungsstoff**

1. Du kannst den Begriff des *Datentyps* beschreiben.
2. Du kannst drei elementare Datentypen von Python aufzählen.
3. Du kannst den Begriff der *Schnittstelle* beschreiben.
4. Du kannst die Schnittstelle des in Python-Datentyps `list` für folgende Operationen verwenden und deren Laufzeitkomplexität $O(\dots)$ angeben. `L`, `L1`, `L2` sind Listen, `i` ist ein gültiger Index und `e` steht für ein Element:
   - Zugriff auf das i-te Element einer Liste: `L[i]` $[O(1)]$
   - Überschreiben des i-ten Elements einer Liste mit `e`: `L[i]=e` $[O(1)]$
   - Hinzufügen von `e` am Ende der Liste: `L.append(e)` $[O(1)]$
   - Entfernen des Elements am Ende der Liste: `L.pop()` $[O(1)]$
   - Entfernen eines Elements an einer Position $0 \leq i < n-1$: `L.pop(i)` $[O(n)]$
   - Einfügen von `e` an einer Position $0 \leq i < n-1$: `L.insert(i,e)` $[O(n)]$
   - Anzahl Elemente von `L`: `len(L)` $[O(1)]$
   - Iterieren über `L`: `for e in L:` `<Anweisung(en)>` $[O(n)]$
   - Verketten von zwei Listen: `L1 + L2` $[O(n_1 + n_2)]$
5. Du kannst die Schnittstelle des in Python-Datentyps `dict` für folgende Operationen verwenden und deren Laufzeitkomplexität $O(\dots)$ angeben. `D` ist ein Dictionary, `k` ein gültiger Schlüssel (*key*) und `v` ist ein Wert (*value*).
   - Zugriff auf das Element mit dem Schlüssel `k`: `D[k]` $[O(1)]$
   - Hinzufügen des Wertes `v` zum Dictionary: `D[k] = v` $[O(1)]$
   - Entfernen des Wertes mit dem Schlüssel `k`: `D.del(k)` $[O(1)]$
   - Prüfen, ob der Schlüssel `k` im Dictionary vorkommt: `k in D` $[O(1)]$
   - Anzahl Elemente in `D`: `len(D)` $[O(1)]$
   - Iterieren über `D`: `for key in D:` `<Anweisung(en)>` $[O(n)]$
6. Du kannst die Schnittstelle des in Python-Datentyps `set` (Menge) für folgende Operationen verwenden und deren Laufzeitkomplexität $O(\dots)$ angeben. (`S`, `S1`, `S2` sind Sets und `e` ein Element.)
   - Das Element `e` zu `S` hinzufügen: `S.add(e)` $[O(1)]$
   - Das Element `e` aus `S` entfernen: `S.remove(e)` $[O(1)]$
   - Prüfen, ob `e` ein Element der Menge `S` ist: `e in S` $[O(1)]$
   - Anzahl Elemente von `S`: `len(S)` $[O(1)]$
   - Iterieren über `S`: `for e in S:` `<Anweisung(en)>` $[O(n)]$

- Vereingungsmenge $\texttt{S1} \cup \texttt{S2}$: `S1.union(S2)` $[O(n_1 + n_2)]^*$
- Schnittmenge $\texttt{S1} \cap \texttt{S2}$: `S1.intersection(S2)` $[O(\min(n_1, n_2))]^*$
- Differenzmenge $\texttt{S1} \setminus \texttt{S2}$: `S1.difference(S2)` $[O(n_1)]^*$

* Diese Laufzeitkomplexitäten werden nicht an der Prüfung verlangt.

7. Abstrakter Datentyp *Stack*:

   - Du kannst die für den Datentyp zentralen Methoden `push(element)`, `pop()`, `peek()`, `size()` und `isEmpty()` aufzählen und in Python auf der Grundlage einer Liste implementieren.
   - Du kannst die Kurzfomrel des Datentyps *Last In – First Out (LIFO)* angeben.
   - Du kannst mindestens drei verschiedene Anwendungen für Stacks aufzählen.
   - Du kannst die Postifix-Darstellung eines arithmetischen Terms in die entsprechende Infix-Darstellung umwandeln und umgekehrt.
   - Du kannst den Verlauf von Operationen auf einem Stack nachvollziehen.

8. Abstrakter Datentyp *Queue*:

   - Du kannst die für den Datentyp zentralen Methoden `enqueue(element)`, `dequeue()`, `size()` und `isEmpty()` aufzählen und in Python auf der Grundlage einer Liste implementieren.
   - Du kannst die Kurzformel des Datentyps *First In – First Out (FIFO)* angeben.
   - Du kannst mindestens drei verschiedene Anwendungen für Queues aufzählen.
   - Du kannst den Verlauf von Operationen auf einer Queue nachvollziehen.

9. Abstrakter Datentyp *Deque*:

   - Du kannst die Datenstruktur *Deque* beschreiben.
   - Du kannst die genaue Bedutung der Abkürzung „*Deque*“ erklären.
   - Du kannst die für den Datentyp zentralen Methoden `addFront()`, `removeFront()`, `addRear()`, `removeRear()`, `size()`, `isEmpty()` aufzählen und in Python auf der Grundlage einer Liste implementieren.
   - Du kannst auf der Grundlage einer Deque eine Funktion schreiben, die überprüft, ob eine Zeichenkette ein Palindrom ist.

10. Abstrakter Datentyp *Linked List* (einfach verkettete Liste):

    - Du kannst die Datenstruktur einer *einfach verketteten Liste* aus einem `Head` und einer Folge von `Node`-Objekte graphisch darstellen (siehe Theorie, 2.4).
    - Du kannst die Elemente einer einfach verketteten Liste aus einem Speicherabbild herauslesen (siehe Aufgabe 3.1 [→ 2.11]).
    - Du kannst die für den Datentyp zentralen Methoden `isEmpty()`, `add(item)`, `size()`, `remove(item)`, `search(item)`, `isEmpty()` aufzählen und die Laufzeitkomplexität $O(\dots)$ dieser Methoden angeben.

- Du kannst das Einfügen eines Elements [`add(item)`] in die einfach verkettete Liste am graphischen Modell und am Speicherabbild darstellen (siehe Aufgabe 3.2 [→ 2.12]).
- Du kannst das Entfernen eines Elements [`remove(item)`] aus der einfach verketteten Liste am graphischen Modell und am Speicherabbild darstellen (siehe Aufgabe 3.3 [→ 2.13]).
- Du kannst die Bedeutung des Begriffs *Garbage Collection* in der Informatik erklären.

11. Abstrakter Datentyp *Tree*:

    - Du kannst drei Informatik-Anwendungen der Datenstruktur `Tree` aufzählen.
    - Du kannst die Begriffe *Knoten*, *Kante*, *Kind(knoten)*, *Eltern(knoten)*, *Geschwister(knoten)*, *Blatt*, *innerer Knoten Pfad*, *Teilbaum*, *Tiefe eines Knotens*, *Höhe eines Baums*, *Binärbaum*, *n-är-Baum*, *Wald* (oder ihre englischen Versionen) den entsprechenden Objekten in der graphischen Darstellung von Bäumen zuordnen.
    - Du kannst den Begriff des *Baums* rekursiv definieren (siehe Theorie, 3.2).
    - Du kannst die folgenden Typen von Binärbäumen an ihrer graphischen Darstellung erkennen: *entarteter Binärbaum*, *voller Binärbaum*, *balancierter Binärbaum*, *vollständiger Binärbaum*, *perfekter Binärbaum*
    - Du kannst einfache binäre oder ternäre Bäume als verschachtelte Listen darstellen.
    - Du kannst binäre Bäume, analog zu den verketteten Listen, durch Knoten und Referenzen (auf allfällige weitere Knoten) darstellen und die Werte aus einem Speicherabbild heraus lesen (siehe Theorie, 3.5).
    - Du kannst Binärbäume wie folgt traversieren: *Preorder*, *Inorder*, *Postorder*

12. Abstrakter Datentyp *Heap (Min-Heap oder Max-Heap)*:

    - Du kannst Listen (ohne „nulltes“ Element) als vollständigen Binärbaum (Heap) darstellen und kannst die Knoten des Baums mit den entsprechenden Indizes der Liste identifzieren.
    - Du kannst überprüfen, ob ein Heap die Min-Heap-Eigenschaft (oder Max-Heap-Eigenschaft) hat.
    - Du kannst Knoten, welche die Min-Heap-Eigenschaft verletzen, durch die Operationen `swim()` und `sink()` in der graphischen Darstellung an die richtige Position „tauschen“.
    - Du kannst im graphischen Modell Elemente zu einem Heap hinzufügen und (falls nötig) die Min-Heap-Eigenschaft durch geeignete Operationen wieder herstellen.
    - Du kannst im graphischen Modell Elemente von der Wurzel eines Heaps entfernen und sowohl die Heap-Struktur als auch die Min-Heap-Eigenschaft durch geeignete Operationen wieder herstellen.