Python (Kapitel 7: Listen) **Prüfungsstoff**

1. Du kannst Listen an ihrer Syntax erkennen: Ein Paar eckiger Klammern, mit Werten, die jeweils durch Kommas getrennt sind. *Beispiel:* `[4, True, 'hello']`
2. Du weisst, dass die Indizierung (Nummerierung) von Python-Listen bei 0 beginnt.
3. Du kannst mit dem Klammeroperator `L[...]` auf die Elemente einer Liste `L` zugreifen. *Beispiel:* `L[3]` gibt das Element mit dem Index 3 (also das 4. Element) zurück.
4. Du kannst einer Liste an einer bestimmten Position einen neuen Wert zuordnen. *Beispiel:* `L[3] = 7` überschreibt das Element mit dem Index 3 mit dem Wert 7.
5. Du kannst erkennen, ob ein Listenzugriff an einer ungültigen Position stattfindet und weisst, dass dies einen `IndexErrror` verursacht.
6. Du kannst mit negativen Indizes von rechts her auf die Listenelmente zugreifen.
7. Du kannst mit Elemente mit mehrfachen Indizies (wie `L[3][2]`) auf Elemente in verschachtelten Listen (`[[...], [...], [...]]`) zugreifen.
8. Du kannst Teillisten mit der Slice-Syntax auswählen:
   - `L[i:j]` Elemente von und mit `i` bis und ohne `j`
   - `L[:j]` Elemente von und mit `0` bis und ohne `j`
   - `L[i:]` Elemente von und mit `i` bis ganz zum Listenende
   - `L[:]` Alle Elemente von `L` („echte“ Kopie der Liste `L`)
   - `L[i:j:s]` Alle Elemente an den Positionen `i`, `i+s`, `i+2s`, ... auswählen, solange diese kleiner als `j` sind (`s`: Schrittweite).
   - `L[::-1]` Kopie der Liste mit Elementen in umgekehrter Reihenfolge.
9. *Listenfunktionen:*
   - Du weisst, dass `len(L)` die Anzahl der Elemente einer Liste zurückgibt.
   - Du weisst, dass `sum(L)` die Summe der Listenelemente zurückgibt, sofern es sich um Zahlen handelt.
10. *Listenoperatoren:*
    - Du weisst, wie Python zwei Listen addiert
      Beispiel: `[1, 2] + [3, 4, 5]` ⇒ `[1, 2, 3, 4, 5]`
    - Du weisst, wie Python eine ganze Zahl mit einer Liste multipliziert.
      Beispiel: `2*[3, 4, 5]` ⇒ `[3, 4, 5, 3, 4, 5]`

11. *Listenmethoden:*

    - Du weisst, dass `L.append(`*`x`*`)` das Element *x* am Ende einer Liste anfügt
    - Du weisst, dass `L.pop()` das letzte Element vom Ende der Liste `L` entfernt und es als Wert zurückgibt, so dass man es bei Bedarf speichern kann.
    - Du weisst, dass `L.pop(`*`i`*`)` das *i*-te Element der Liste `L` entfernt und es als Wert zurückgibt, so dass man es bei Bedarf speichern kann.
    - Du weisst, dass `L.insert(`*`i, elmt`*`)` das Element *elmt* an der Position *i* der Liste `L` einfügt und alle davon betroffenen Elemente um eine Position nach rechts verschiebt.
    - Du weisst, dass `L.sort()` die Elemente der Liste *inplace* sortiert; d. h. die alte Reihenfolge mit der sortierten Liste überschreibt.
    - Du weisst, dass `L.reverse()` die Reihenfolge der Elemente der Liste `L` *inplace* umkehrt.

12. Du kannst Mehrfachzuweisung mit Listen interpretieren.

    Beispiel: `x, y, z = [4, 1, 7]` ⇒ `y=1`

13. Du kannst Listen mit `for` elementweise durchlaufen. *Beispiel:*

```
L = [3, 7, 5, 1]
for x in L:
    print(x)      # => 3
                  # => 7
                  # => 5
                  # => 1
```

14. Du kannst eine Liste indexgesteuert durchlaufen und bestimmte Elemente herausfiltern. *Beispiel:*

```
L = [3, 7, 5, 1]
n = len(L)
for i in range(n)
    if L[i] > 4:
        print(L[i])  # 7
                     # 5
```

15. Du kannst einfache List Comprehensions interpretieren. List Comprehensions sind Listen, die mit einer for-Schleife „von innen“ erzeugt werden. *Beispiele:*

    - `L = [0 for i in range(0, 5)]`
      `print(L)` ⇒ `[0, 0, 0, 0, 0]`
    - `L = [10*k for k in range(2, 5)]`
      `print(L)` ⇒ `[20, 30, 40]`
    - `L = [[i for j in range(0,3)] for i in range(1,4)]`
      `print(L)` ⇒ `[[1, 1, 1], [2, 2, 2], [3, 3, 3]]`