1. Du kannst in Python Listen mit `[..., ..., ...]` definieren.
2. Du kannst für eine gegebene Liste `L` angeben, welches Element mit der Indexschreibweise `L[i]` ausgewählt wird. Darüber hinaus kannst du auch negative Indizes interpretieren und feststellen, ob ein `IndexError` auftritt.
3. Du kannst nachvollziehen, wie eine gegebene Liste `L` durch Zuweisungen verändert wird. Auch hier wird das Verständnis für die Indexschreibweise vorausgesetzt.
4. Du kannst mit mehreren Indizes auf die Elemente verschachtelter Listen zugreifen.
5. Du kannst angeben, welche Teilliste mit der Slice-Syntax `L[i:j]` aus einer Liste ausgewählt wird und kannst auch mit den Spezialfällen `L[:j]`, `L[i:]` und `L[:]` umgehen.
6. Du weisst, dass bei der Zuweisung einer Liste zu einer Variablen (grob gesagt) nur der Speicherort des ersten Elements und die Anzahl der Elemente in der Variable gespeichert werden. Dies führt dazu, dass beim „Kopieren“ einer Listenvariable nur eine weitere Referenz auf dieselbe Liste erzeugt wird und Veränderungen an beiden Variablen an der gleichen Liste durchgeführt werden.
7. Du weisst, wie man echte Kopien einer Liste mit der Slice-Syntax erzeugt.
8. Syntax und Semantik der folgenden Listenoperatoren:
   - „Addition“ von Listen mit `"+"`
   - „Multiplikation“ von Listen mit `"*"`
   - Testen der Listenzugehörigkeit von Elementen mit `"in"` und `"not in"`.
9. Syntax und Semantik der folgenden Listenfunktionen:
   - `len(L)`: Liefert die Anzahl der Elemente der Liste `L` zurück.
   - `sorted(L)`: Liefert eine Liste mit aufsteigend sortierten Elementen zurück.
   - `sorted(L, reverse=True)`: Liefert absteigend sortierte Liste zurück.
10. Syntax und Semantik der folgenden Listenmethoden
    - `L.append(x)`: Fügt `x` am Ende von `L` hinzu.
    - `L.insert(i, x)`: Verschiebt die Elemente mit dem Index `i`, `i+1`, ... um jeweils eine Position nach rechts und fügt `x` an der Position `i` ein.
    - `L.pop()`: Entfernt das letzte Element von `L` und liefert es als Wert zurück
    - `L.pop(i)`: Entfernt das `i`-te Element aus `L` und liefert es als Wert zurück

11. Du kannst Listen mit einer elementgesteuerten `for`-Schleife verarbeiten. Zum Beispiel alle Elemente einer Liste addieren:

```
    summe = 0
    for x in L:
        summe = summe + x
```

12. Du kannst Listen mit einer indexgesteuerten `for`-Schleife verarbeiten. Zum Beispiel alle Elemente einer Liste verdoppeln:

```
    for i in range(0, len(L)):
        L[i] = 2*L[i]
```

13. Du kannst Listen mit der `enumerate()`-Funktion und einer `for`-Schleifen verarbeiten. Zum Beispiel alle Elemente einer Liste verdoppeln:

```
    for i, x in enumerate(L):
        L[i] = 2*x
```

14. Du kannst Schleifen mit `break` abbrechen oder einzelne Schleifendurchläufe mit `continue` überspringen.