Listen

- Datentypen, welche mehrere Werte in einer Variable speichern können
- Elemente = Werte der Liste
- Andere Programmiersprachen: Arrays (aber gleiche Funktionalität)
- Werte können unterschiedliche Datentypen haben
- Anzahl der Elemente ist flexibel erweiterbar

Listen erstellen

- Syntax: Eckige Klammern!

```
list_1 = []                 # leere Liste
list_2 = [5, 6, 7]          # Liste mit 3 Werten vom Typ Integer
list_3 = ["x", "y"]         # Liste mit 2 Werten vom Typ String
list_4 = [9, "abc", 23.5]   # Liste mit 3 Werten unterschiedlicher Datentypen

# auch möglich: Listen in Listen! (Dazu später mehr…)
list_5 = [[1, 2, 3], [True, False, True]]
```

Auf Listenelemente zugreifen

- Jedes Listenelement hat einen zugehörigen, einzigartigen Index
- Beispiel: my_list = ["a", 7.9, 65732]

Index	**0**	**1**	**2**
Wert	"a"	7.9	65732

- Der Index startet **immer** bei 0 und läuft hoch (Ganzzahl)
- Regel: Eine Liste mit x Elementen hat die Indizes 0 bis x – 1

Auf Listenelemente zugreifen

- Jedes Listenelement hat einen zugehörigen, einzigartigen Index
- Beispiel: my_list = ["a", 7.9, 65732]

Index	**0**	**1**	**2**
Wert	"a"	7.9	65732

- Auf Listenelemente zugreifen: Über den Index

```
print(my_list[0])     # Ausgabe: "a"
print(my_list[2])     # Ausgabe: 65732
var_x = my_list[1]    # Erstelle Variable mit Wert 7.9
print(my_list[3])     # IndexError (es gibt kein Element mit Index 3)
```

Auf Listenelemente zugreifen: Negativer Index

- Jedes Listenelement hat einen zugehörigen, einzigartigen Index
- Beispiel: my_list = ["a", 7.9, 65732]

Index	**0**	**1**	**2**
Wert	"a"	7.9	65732

- Auf Listenelemente zugreifen: Mit negativem Index
- Wenn die Anzahl der Elemente variabel ist (Liste als Parameter z.B.)

```
print(my_list[-1])    # Ausgabe: 65732 (das letzte Element)
print(my_list[-3])    # Ausgabe: "a"
print(my_list[-4])    # IndexError
```

- Was passiert intern?

Auf Listenelemente zugreifen: Slice

- Beispiel: my_list = ["a", 7.9, 65732, True, 23.1]
- Per Slice mehrere Elemente aus der Liste holen („rausschneiden“)
- Syntax: [a:b] (a = Anfang, b = exklusives Ende) (a & b sind optional!)

```
print(my_list[1:3])  # Ausgabe: [7.9, 65732] (Beachte: ohne my_list[3]!)
print(my_list[:4])   # Ausgabe: ["a", 7.9, 65732, True]
print(my_list[3:])   # Ausgabe: [True, 23.1]
print(my_list[:])    # Ausgabe: ["a", 7.9, 65732, True, 23.1]
```

- Lässt sich der letzte Befehl noch kürzer schreiben?

Listenelemente verändern

- Ablauf: Listenelement über Index adressieren -> neuen Wert zuweisen
- Beispiel: gerade_zahlen = [2, 4, 7, 8]

```
print(gerade_zahlen)       # Ausgabe: [2, 4, 7, 8]
gerade_zahlen[2] = 6       # Verändert 7 zu 6
print(gerade_zahlen)       # Ausgabe: [2, 4, 6, 8]
```

- Mehrere Elemente verändern: ungerade_zahlen = [1, 4, 6, 7]

```
ungerade_zahlen[1:3] = [3, 5]    # Verändert [4, 6] zu [3, 5]
print(ungerade_zahlen)    # Ausgabe: [1, 3, 5, 7]
```

Listenelemente hinzufügen

- Liste um Elemente erweitern: .append(x) (x ist ein beliebiger Wert)
- Liste um Listenelemente erweitern: .extend(y) (y ist eine Liste, o.Ä)
- Beispiel: gerade_zahlen = [2, 4, 6, 8]

```
gerade_zahlen.append(10)          # Hängt die 10 ans Ende
print(gerade_zahlen)              # Ausgabe: [2, 4, 6, 8, 10]
gerade_zahlen.extend([12, 14])    # Hängt 12 und 14 ans Ende
print(gerade_zahlen)              # Ausgabe: [2, 4, 6, 8, 10, 12, 14]
```

- Statt .extend() auch + (Additionsoperator) möglich

Listenelemente hinzufügen

- Neue Elemente an einer bestimmten Position hinzufügen:
- Mit .insert(idx, z)
  - idx gibt die Position (also den Index) an, wo z hinzugefügt werden soll
  - z ist ein beliebiger Wert, der hinzugefügt werden soll
  - Anmerkung: Es wird „Platz gemacht“ für den neuen Wert (kein Überschreiben)
- Beispiel: ungerade_zahlen = [1, 5, 7]

```
ungerade_zahlen.insert(1, 3)     # Fügt an Index 1 den Wert 3 hinzu
print(ungerade_zahlen)           # Ausgabe: [1, 3, 5, 7]
```

Listenelemente löschen

- Listenelemente löschen: Mehrere Wege
- Beispiel: gerade_zahlen = [2, 3, 4, 5, 6]
- Über den Index löschen: 2 Möglichkeiten

```
del gerade_zahlen[1]        # entfernt 3 (Achtung: Verändert Länge!)
print(gerade_zahlen)        # Ausgabe: [2, 4, 5, 6]
gerade_zahlen.pop(2)        # entfernt 5
print(gerade_zahlen)        # Ausgabe: [2, 4, 6]
```

- Anmerkung: .pop() ohne Parameter: Entferne das letzt Element

Listenelemente löschen

- Auch möglich: Elemente anhand des Werts löschen
- Mit .remove(a) (a ist der Wert, der gelöscht werden soll)
- Achtung:
  - Setzt voraus, dass der Wert existiert! Sonst Fehler
  - Löscht den Wert nur 1 Mal
- Beispiel: namen_mit_a = ["Aylin", "Bertha", "Anton", "Bertha"]

```
namen_mit_a.remove("Bertha")
print(namen_mit_a)              # Ausgabe: ["Aylin", "Anton", "Bertha"]
namen_mit_a.remove("Bertha")
print(namen_mit_a)              # Ausgabe: ["Aylin", "Anton"]
namen_mit_a.remove("Bertha")    # ValueError! Kann "Bertha" nicht finden
```

Weitere praktische Listenmethoden

- Was sind Methoden? Funktionen einer Klasse (-> Duck Typing)
- Alle Elemente einer Liste entfernen: .clear()
- Index eines Elementes ausgeben: .index(elem)
  - Achtung: Gibt den Index des ersten gefunden Elementes aus
- Zählen, wie oft ein Element vorkommt: .count(elem)
- Liste sortieren (aufsteigend): .sort()
  - Sortieren ist ein recht komplexes Thema bei gemischten oder String-Listen
- Reihenfolge der Listenelemente umkehren: .reverse()
- Liste kopieren: .copy()

Listen können noch mehr!

- Länge einer Liste ausgeben: len(list)
  - Ist keine Listenmethode!
  - Gibt Anzahl der Elemente aus, nicht höchsten Index!
- Prüfen ob ein Element in einer Liste vorkommt: mit if-Verzweigung
- Beispiel: x = [1, 2, 3]

```
if 5 in x:
    x.remove(5)
else:
    print("Konnte 5 nicht finden")
# Ausgabe: "Konnte 5 nicht finden"
```

Ihr habt‘s geschafft! 😹😹😹

- Hausaufgaben: 07 – Hausi.py