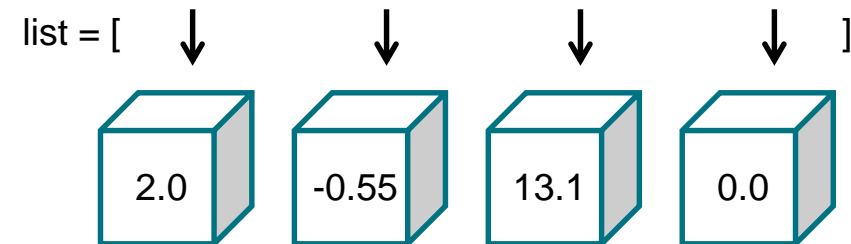


- Tupel können nicht geändert werden. Es kann weder ein Wert geändert noch ein Wert hinzugefügt werden.
- Für viele Anwendungen ist das unpraktisch. Zu einer Einkaufsliste beispielsweise sollte man etwas hinzufügen können oder etwas daraus entfernen. (Sonst müsste man ja immer das Gleiche einkaufen.)

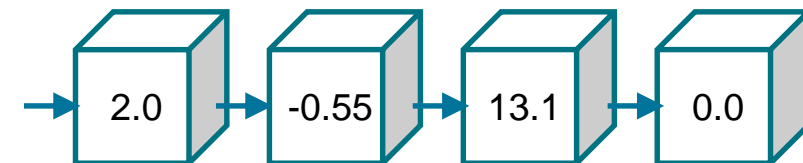
- In Programmiersprachen gibt es dafür Listen.

- Eine **Liste** in Python besteht aus einer beliebigen, veränderlichen Anzahl von Referenzen. Beispiel:

```
list = [2.0, -0.55, 13.1, 0.0]
```



- Die Referenzen können auf beliebige Werte zeigen, auch auf Werte unterschiedlichen Datentyps.
- Listen können mit print ausgegeben werden.
- Man kann sich eine Liste auch als Verkettung von Elementen mit einem Anfang und einem Ende vorstellen



- Wie kann man nun auf die einzelnen Elemente einer Liste zugreifen?

- a) Entpacken geht zwar, macht aber wegen der veränderlichen Länge einer Liste normalerweise keinen Sinn.

- b) Indexzugriff, die Elemente sind mit den Indices 0, 1, 2, ... durchnummeriert. Beispiel:

```
list = [2.0, -0.55, 13.1, 0.0, -1.5]  
print(list[1])
```

Auch hier ist die Nummerierung $-1, -2, -3, \dots$ von rechts möglich.

- c) In einer Schleife. Beispiel:

```
list = [2.0, -0.55, 13.1, 0.0, -1.5]  
for i in list:  
    print(i)
```

- Mit der Funktion `len` kann man die Länge einer Liste herausfinden. Beispiel:

```
list = [2.0, -0.55, 13.1, 0.0, -1.5]
print(len(list))                # Ausgabe: 5
```

- Wie kann man Listen ändern?

- a) Änderung eines Wertes durch Indexzugriff. Beispiel:

```
list[1] = "neu"
```

Falls der Index größer oder gleich der Länge der Liste ist, gibt es einen Fehler.

- b) Hinzufügen eines Wertes am Ende durch `append`. Beispiel:

```
list.append("am Ende")
```

- c) Einfügen eines Wertes an eine bestimmte Position durch `insert`. Die vorhandenen Elemente werden nach hinten verschoben. Beispiel:

```
list.insert(1, "ganz neu")
```

Falls die Position größer oder gleich der Länge ist, wird der Wert hinten an die Liste angehängt.

- d) Löschen eines Wertes durch `remove`. Das erste Vorkommen des Wertes wird gelöscht. Beispiel:

```
if "neu" in list:  
    list.remove("neu")
```

Falls der Wert nicht enthalten ist, gibt es einen Fehler.

- Das folgende Beispiel zeigt die Funktionen.

```
list = [2.0, -0.55, 13.1, 0.0, -1.5]
```

```
print(list)
```

```
# Ausgabe: [2.0, -0.55, 13.1, 0.0, -1.5]
```

```
list[1] = "neu"
```

```
print(list)
```

```
# Ausgabe: [2.0, 'neu', 13.1, 0.0, -1.5]
```

```
list.append("am Ende")
```

```
print(list)
```

```
# Ausgabe: [2.0, 'neu', 13.1, 0.0, -1.5, 'am Ende']
```

```
if "neu" in list:
```

```
    list.remove("neu")
```

```
print(list)
```

```
# Ausgabe: [2.0, 13.1, 0.0, -1.5, 'am Ende']
```

```
list.insert(1, "ganz neu")
```

```
print(list)
```

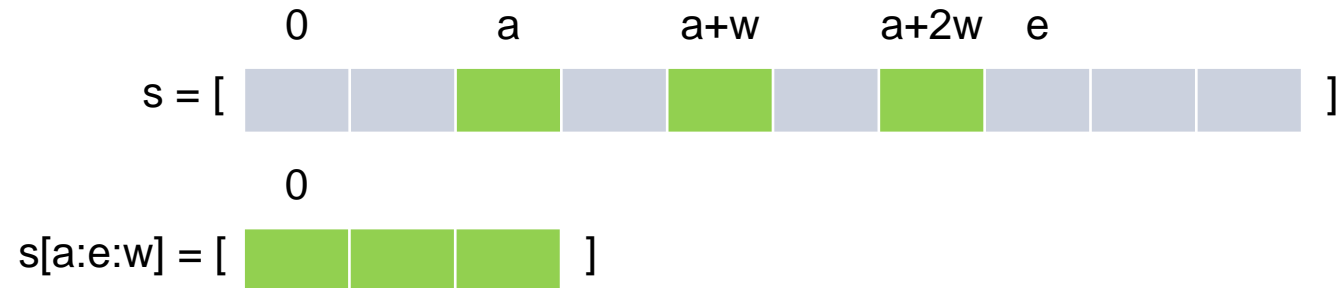
```
# Ausgabe: [2.0, 'ganz neu', 13.1, 0.0, -1.5, 'am Ende']
```

- Ein häufig vorkommendes Anwendungsbeispiel für Listen sind Adressbücher.
- Die Listenelemente sind Kontakte (Tupel).
- Die Anzahl der enthaltenen Kontakte ist beliebig.
- Man muss Kontakte hinzufügen, Kontakte ändern, Kontakte entfernen und Kontakte suchen können.
- Beispiel für das Suchen aller Kontakte mit dem Nachnamen name (zur Vereinfachung sind Kontakte auf Name, Vorname verkürzt):

```
adressen = [ ("Mustermann", "Max"), ("Musterfrau", "Smilla"), ("Mustermann", "Emil") ]  
name = "Mustermann"
```

```
for kontakt in adressen:  
    if kontakt[0] == name:  
        print(kontakt)
```

- Wir haben das Slicing schon bei Zeichenketten besprochen. Allgemein kann Slicing auf alle sequentiellen Datenstrukturen wie Zeichenketten, Tupel oder Listen angewandt werden.
- $s[a : e : w]$ ergibt $s[a]$, $s[a + w]$, $s[a + 2*w]$, ... $s[a + i*w]$, falls i die größte Zahl ist, sodass $a + i*w < e$.



■ Beispiele:

```
list = [2.0, -0.55, 13.1, 0.0, -1.5, 1.5, 2.0]
```

```
print(list[1:5])
```

Ausgabe: [-0.55, 13.1, 0.0, -1.5]

```
print(list[1:5:2])
```

Ausgabe: [-0.55, 0.0]

```
print(list[1:])
```

Ausgabe: [-0.55, 13.1, 0.0, -1.5, 1.5, 2.0]

```
print(list[:6])
```


Ausgabe: [2.0, -0.55, 13.1, 0.0, -1.5, 1.5]

```
print(list[5:1:-2])
```

Ausgabe: [1.5, 0.0]

- Neben den eben gezeigten Funktionen gibt es noch einige weitere Funktionen. Der Vollständigkeit halber zeigt die folgende Liste auch die bereits bekannten Funktionen:

<code>list[i]</code>	Indexzugriff	<code>list += list2</code>	Anhängen
<code>list.append(x)</code>	Anhängen	<code>list.count(x)</code>	Bestimmen der Anzahl
<code>list.insert(i, x)</code>	Einfügen	<code>list.index(x)</code>	Positionsbestimmung
<code>x in list</code>	Enthaltensein	<code>list.index(x, i, j)</code>	Positionsbestimmung
<code>list.remove(x)</code>	Entfernen	<code>list[a:e:w]</code>	Slicing
<code>list.pop()</code>	Entfernen	<code>len(list)</code>	Länge
<code>list.pop(i)</code>	Entfernen	<code>sum(list)</code>	Summe der Elemente
<code>list.extend(list2)</code>	Anhängen	<code>min(list)</code>	Minimum der Elemente
<code>list = list + list2</code>	Anhängen	<code>max(list)</code>	Maximum der Elemente

- Bitte lesen Sie die Details in (Klein 2018) nach. Am besten probieren Sie die Funktionen an selbst gewählten Beispielen aus. Verwenden Sie z.B. Listen von Zeichenketten, Listen von Tupel, Listen von Listen.
-  Kapitel 5.1.2, 16.1 – 16.10 in (Klein 2018)



1. „Schreiben Sie eine Funktion, die die Reihenfolge einer Liste umkehrt. Verzichten Sie aber auf die Benutzung der Listen-Methode ‚reverse‘ und versuchen Sie stattdessen, nur die Methoden ‚pop‘ und ‚append‘ zu benutzen.“ (Klein 2018, Seite 155).

```
def rev(lst):
```

Parameter: list lst

Rückgabewert: list

Beispiel:

```
liste = ["a","b","c","d"]
```

```
liste = rev(liste)
```

```
print(liste)           # Ausgabe: ['d', 'c', 'b', 'a']
```

2. Noch flexibler wäre das Beispiel Brüche, wenn die Eingabe vor der Verarbeitung in eine Liste abgespeichert würde, z.B. $-5/4 + 7/6 : 2/12$ in die Liste `[(-5, 4), "+", (7, 6), ":", (2, 12)]`. Ändern Sie das Beispiel entsprechend.



3. Im Kapitel über Bedingte Anweisungen hatten wir in einer Aufgabe das Ablaufdiagramm für folgenden Algorithmus erstellt:

Algorithmus sort:

Eingabe: Zahlen z_1, z_2, \dots, z_n

Ausgabe: Die Zahlen in sortierter Reihenfolge, kleinste Zahl zuerst

Jetzt können wir diesen Algorithmus in Python implementieren. Erstellen Sie eine Funktion

```
def sort(z):
```

Parameter: list z : Liste von Zahlen

Rückgabewert: Keiner, die Liste z wird sortiert

Verwenden Sie dabei for-Schleifen. (Falls Sie bereits wissen, dass es in Python bereits eine Sortierfunktion gibt: Wir besprechen die „eingebaute“ Sortierfunktion im nächsten Kapitel.)



4. In der Sortierfunktion müssen zwei Listeneinträge vertauscht werden. In den meisten Programmiersprachen verwendet man dazu eine Hilfsvariable.

Sollen ganz allgemein die Werte der Variablen a und b vertauscht werden, kann man das folgendermaßen programmieren:

```
temp = a
a = b
b = temp
```

In Python gibt dafür natürlich eine Abkürzung:

```
a, b = b, a
```

Ändern Sie die Sortierfunktion entsprechend.