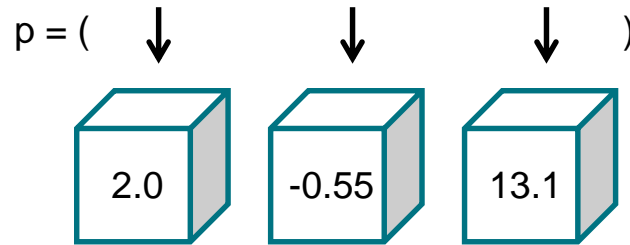


- Im Praktikum verwenden wir Punkte (oder Vektoren) im Raum. Ein Punkt wird durch drei Variablen  $x, y, z$  repräsentiert. In der Mathematik kennt man dafür auch die Schreibweise  $p = (x, y, z)$ . Wir haben also statt drei Variablen  $x, y, z$  nur eine Variable  $p$ . Dadurch können Programme vereinfacht werden.
- Ein weiteres Anwendungsbeispiel für Tupel sind Kontakteinträge:
  - Jeder Kontakteintrag folgt dem selben Muster. An erster Stelle kommt der Familienname, dann ...
  - Die Anzahl der Elemente ist in jedem Eintrag gleich. Falls ein Eintrag fehlt, wird ein leeres Datenfeld eingetragen.
  - Beispiel:

("Mustermann", "Max", "", "Esplanade", "10", "85049", "Ingolstadt")

- Ein **Tupel** in Python besteht aus einer beliebigen, aber festen Anzahl von Referenzen.



- Die Anzahl der Referenzen kann nachträglich nicht mehr geändert werden.
- Die Referenzen können auf beliebige Werte zeigen, auch auf Werte unterschiedlichen Datentyps.
- Die runden Klammern können auch weggelassen werden.
- Tupel können mit `print` ausgegeben werden.

## ■ Beispiele:

```
vector = (2.0, -0.55, 13.1, 0.0, -1.5)
print(vector)
person = ("Mustermann", "Max", 29, 2, 2019)
print(person)
person = ("Mustermann", "Max", (29, 2, 2019))
print(person)
person = "Mustermann", "Max", (29, 2, 2019)
print(person)
```

- Ein Tupel kann auch wieder ein Tupel beinhalten. Im Beispiel ist das Geburtsdatum als drei Zahlen gespeichert oder als Tupel.

- Wie kann man nun auf die einzelnen Elemente eines Tupels zugreifen?

a) Entpacken. Beispiel:

```
p = (2.0, -0.55, 13.1)
(x, y, z) = p
print(y)
```

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

```
p = (2.0, -0.55, 13.1)
print(p[1])
```

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

Wem das Tippen der eckigen Klammern bei häufiger Verwendung von `p[1]` zu umständlich ist, kann folgende Abkürzung einführen:

```
x = p[1]
```

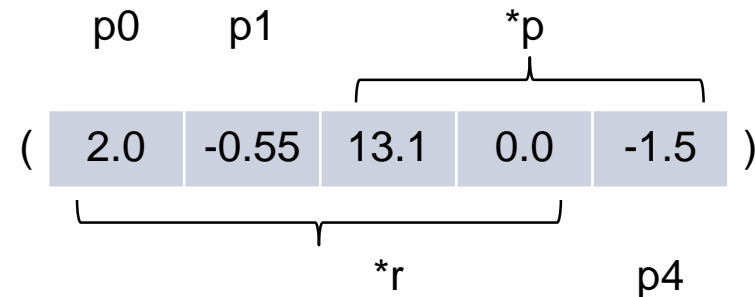
### c) In einer Schleife. Beispiel:

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

### d) Das Entpacken, wie oben gezeigt, benötigt so viele Variablen, wie das Tupel Elemente hat. Kürzer geht es folgendermaßen. Beispiel:

```
p = (2.0, -0.55, 13.1, 0.0, -1.5)
(p0, p1, *q) = p
print(p1)

(*r, p4) = p
print(p4)
```




`*q` und `*r` ersetzen die fehlenden Variablen. (Genauer: `q` und `r` sind Listen, der fehlenden Variablen. Listen behandeln wir im Anschluss an dieses Kapitel.)

- Einzelne Elemente von Tupel können nicht geändert werden. Beispiel:

```
p = (2.0, -0.55, 13.1)
p[1] = 5.0           # das geht nicht
p = (p[0], 5.0, p[2]) # so geht's
print(p)
```

Man kann nur das gesamte Tupel ersetzen durch ein neues Tupel. Das ist aber umständlich.

- Tupel sind unveränderlich. In einem späteren Kapitel besprechen wir das genauer.
- Mit Tupel können wird das Beispiel Brüche einfacher (oder zumindest eleganter) schreiben.
  - bruch = (zaehler, nenner)
  - Wir ändern das Programm in der Vorlesung. Die beiden geänderten Skripte finden Sie auf Moodle.
-  Kapitel 5.1.3 in (Klein 2018)



1. Ergänzen Sie das Beispiel Brüche um die Subtraktion und die Division. Verwenden Sie nicht "-" bzw. "/" als Rechensymbole, sondern "~" und ":". Sonst müssten Sie zwischen Rechensymbol und Vorzeichen bzw. Bruchstrich unterscheiden.
2. Ändern Sie das Beispiel Brüche, sodass längere Rechnungen möglich sind, z.B.  $-5/4 + 7/6 : 2/12$ . Punkt vor Strich soll nicht gelten. Lösungsidee:

Schreiben Sie zuerst eine Funktion, die das erste Rechensymbol findet:

```
def find_op(s):
```

Parameter:        str s : Eine Rechnung, z.B.  $-5/4 + 7/6 : 2/12$

Rückgabewert:    tuple : Tupel bestehend aus der Position des ersten Rechensymbols und dem  
Rechensymbol oder das Tupel (-1, ""), falls kein Rechensymbol gefunden wurde

Gehen Sie dazu in einer for-Schleife durch die Rechnung und suchen Sie nach einem Rechensymbol. Wenn das erste Rechensymbol gefunden wurde, return. Nach der Schleife, return (-1, "")

Im Hauptprogramm wird die Rechnung schrittweise ausgeführt:

- 1) Initialisierung des Ergebnisses mit (0, 1) und des Operators mit "+"
- 2) Aufruf der Funktion find\_op
- 3) Falls kein Rechensymbol gefunden wurde, ist der letzte Bruch erreicht. Dieser Bruch wird mit dem Operator auf das Ergebnis „draufgerechnet“ und das Programm ist zu Ende
- 4) Sonst wird der erste Bruch in der Rechnung ebenfalls auf das Ergebnis „draufgerechnet“. Aber danach werden der Bruch und das Rechensymbol von der Eingabe abgespalten.
- 5) Der neue Operator ist das gefundene erste Rechensymbol.
- 6) Weiter mit Schritt 2)

