Die Liste (`list`)

Eine `list` ist in Python ein neuer Datentyp, der aus null (leere Liste) oder mehreren oder ganz vielen Werten besteht. Diese Werte dürfen durchaus verschiedene Datentypen haben (Listen dürfen selbst auch Listen enthalten). Man gibt die Werte in **eckigen Klammern** an:

```
[2, 3.5, True, "Reinhard Stix", [0,1]]
[ int, float, Wahrheitswert, string, list ]
```

Durch Zuweisungen speichert man solche Listen in Variablen:

```
x = [2, 3.5, True, "Reinhard Stix", [0,1]]
type(x)        (<class 'list'>)
```

`+` und `*` funktionieren gleich wie bei Strings (Python realisiert nämlich Strings als Listen!)

```
y = x + [3, 5]        # erzeugt eine aneinandergehängte Liste
y = 2*x               # dasselbe wie y = x + x (duplizierte Liste)

x.append(-100)        # hängt -100 an die Liste x an
```

Auf die einzelnen Elemente eine Liste greift man mit dem Index in eckigen Klammern zu. Indices beginnt man immer ab 0 zu zählen:

```
x[0]                  # das erste Element von x: 2
x[4]                  # das fünfte Element von x: [0,1]
```

Negative Indices sind auch erlaubt und durchlaufen die Elemente von hinten nach vorn:

```
x[-1]                 # das letzte Element von x: -100
x[-2]                 # das vorletzte Element von x: [0,1]

len(x)                # Anzahl der Elemente von x: 6
```

Man darf einzelne (**existierende**!) Listenelemente überschreiben:

```
x[0] = 2000
x[-1] = x[2]
```

Man kann sehr einfach Teil-Listen von Listen erzeugen:

```
x[2:]                 # Liste von x[2] bis zum Ende von x
x[:2]                 # Liste von x[0] bis x[1] (nicht x[2])
x[2:5]                # Liste von x[2] bis x[4] (nicht x[5])
x[2:12:3]             # Jedes 3. Element: [ x[2],x[5],x[8],x[11] ]
x[8:5:-1]             # absteigende Indices: [ x[8],x[7],x[6] ]
```

Bei Teil-Listen gehört der Startindex immer dazu, der Endindex NICHT

Das Tupel (`tuple`)

Ein `tuple` ist eine „nicht veränderbare Liste“. Man gibt die Werte bei der Definition durch Beistriche getrennt an, viele Programmierer umrahmen die Werte mit **runden Klammern**, damit bei komplizierteren Ausdrücken das Tupel gut zu erkennen ist. Alle Indexoperationen funktionieren auch mit Tupeln. Das Anhängen an ein `tuple` (z.B. mittels `append()`) ist natürlich nicht erlaubt, auch nicht das Überschreiben von `tuple`-Elementen! Wenn man etwas sowohl als Liste oder als Tupel modellieren kann, sollte man das Tupel vorziehen.

```
x = 2, 3.5, True, "Reinhard Stix", [0,1]
x = ( 2, 3.5, True, "Reinhard Stix", [0,1] )  # das Gleiche

x + (3, 5)      # ( 2, 3.5, True, "Reinhard Stix", [0,1], 3, 5 )
x + [3, 5]      # Fehler: Man kann list und tuple nicht „addieren“
2 * x
list(x)             # mache aus x eine Liste
tuple([1, 2, 3])    # die andere Richtung

x[0]                # das erste Element von x: 2
x[4]                # das fünfte Element von x: [0,1]
x[-1]               # das letzte Element von x: -100
x[-2]               # das vorletzte Element von x: [0,1]

len(x)              # Anzahl der Elemente von x: 5
x = 1,              # oder x = (1,) : Tupel mit 1 Element (Beistrich ist wichtig)
```

Man kann sehr einfach Teil-`tuple` erzeugen:

```
x[2:]               # Tupel von x[2] bis zum Ende von x
x[:2]               # Tupel von x[0] bis x[1] (nicht x[2])
x[2:5]              # Tupel von x[2] bis x[4] (nicht x[5])
x[2:12:3]           # Jedes 3. Element: ( x[2],x[5],x[8],x[11] )
x[8:5:-1]           # absteigende Indices: ( x[8],x[7],x[6] )
```

Die Menge (`set`)

Der Vollständigkeit halber erwähne ich noch den Python Datentyp `set`, der seltener verwendet wird und in meinen Beispielen gar nicht vorkommt. Die Werte werden hier in `{}` angegeben:

```
obst = { "Apfel", "Birne", "Kirschen" }
```