# 19 Datentyp Tupel

## 19.1 Beschreibung

- ein Tupel ist wie eine Liste eine Sequenz von Elementen
- jedoch sind die einzelnen Elemente nicht änderbar
- damit ist ein Tupel ressourcenschonender als eine Liste
- für Tupel gibt es nur ein Teil der Listen-Funktionalitäten – es gibt nur „lesenden“ Funkonen , es gibt keine modifizierenden Funktionen auf Tupel-Elemente
- ein Tupel kann unterschiedliche Datentypen enthalten, also auch Tupel und Listen
- ein Tupel wird durch die Zuweisung von Elementen innerhalb runder Klammern angelegt
- mit Tupel-Operatoren und Tupel-Funktionen lassen sich Tupel untersuchen
- es lassen sich ein- und mehrdimensionale Felder (Arrays) abbilden
- gleiche Werte können mehrfach vorkommen

- Beispiel:

```
# Tupel anlegen
zahlen = (1, 5, 7, 9, 22, 35, 1)
# Tupel ausgeben
print(zahlen)
```

```
(1, 5, 7, 9, 22, 35, 1)
```

```
# 1. Element des Tupels ausgeben (index fängt bei 0 an)
print(zahlen[0])
```

```
1
```

```
# 2. bis 4. Element des Tupels ausgeben (index = 1:4)
print(zahlen[1:4])
```

```
(5, 7, 9)
```

```
# Tupel-Definition mit Elementen vom Typ int, tuple und list
t= (1, (2,4,6), [12, 17], 20)
print(t)
for i in t:

    print(type(i), i)
```

```
< (1, (2, 4, 6), [12, 17], 20)
<class 'int'> 1
<class 'tuple'> (2, 4, 6)
<class 'list'> [12, 17]
<class 'int'> 20
```

## 19.2 Tupel-Operatoren

- Verketten von Tupeln mit **+**
- Vervielfachen von Tupeln mit *****
- Zugriff auf ein Element per Index (Zählweise beginnt bei 0) mit **[i]**
- Zugriff auf einen Bereich von Elementen per Index (Zählweise beginnt bei 0) mit **[a:b]**
- Prüfen, ob ein Element im Tupel enthalten ist mit **in**
- Prüfen, ob ein Element im Tupel NICHT enthalten ist mit **not in**

- Beispiele:

```
# Tupel verketten
tupel_t1 = ["Das", "ist"]
tupel_t2 = ["eine", "Addition"]
tupel_gesamt = tupel_t1 + tupel_t2
print(tupel_gesamt))
```

```
['Das', 'ist', 'eine', 'Addition']
```

```
# Zugriff auf einen Bereich von Elementen in einem Tupel (mit Index)
print(tupel_gesamt[2:4])
```

```
['eine', Addition]
```

```
# Vervielfachen von Tupeln
tupel_3 = ["Python", "ist", "eine", "Programmiersprache"]
print(tupel_3)
```

```
['Python', 'ist', 'eine', 'Programmiersprache']
```

```
print(2*tupel_3)
```

```
['Python', 'ist', 'eine', 'Programmiersprache', 'Python', 'ist', 'eine',
'Programmiersprache']
```

## 19.3 Länge eines Tupels (Anzahl der Elemente)

- Länge eines Tupels (Anzahl der Elemente) **len(tupel)**

- Beispiele:

```
# Länge eines Tupels ausgeben
zahlen = (1, 2, 3, 5, 7, 9, 12, 67, 78)
print("Länge des Tupels'zahlen': ", len(zahlen))

print(zahlen[1:4])

for i in range(0, len(zahlen)):
    print("Index", i, "Wert", zahlen[i])
```

```
[Länge des Tupels 'zahlen':  9
[2, 3, 5]
Index 0 Wert 1
Index 1 Wert 2
Index 2 Wert 3
Index 3 Wert 5
Index 4 Wert 7
Index 5 Wert 9
Index 6 Wert 12
Index 7 Wert 67
Index 8 Wert 78]
```

## 19.4 Tupel-Funktionen (Methoden)

- Vorkommen eines Elementes im Tupel zählen **.count("xy")**
- Index des ersten Vorkommens eines Elements im Tupel ermitteln **.index("xy")**

- Beispiele:

```
# Vorkommen eines Elementes im Tupel zählen .count("xy")
t_1 = (16, 37, 31, 10, 27)
t_2 = (45, 76, 53, 28, 37)
t_gesamt = t_1 + t_2
print(t_1)
print(t_2)
print("37 kommt", t_1.count(37), "-mal in l_1 vor")
print("37 kommt", t_2.count(37), "-mal in l_2 vor")
print("37 kommt", t_gesamt.count(37), "-mal in l_gesamt vor")
```

```
[16, 37, 31, 10, 27]
[45, 76, 53, 28, 37]
37 kommt 1 -mal in l_1 vor
37 kommt 1 -mal in l_2 vor
37 kommt 2 -mal in l_gesamt vor
```

```
# Index des ersten Vorkommens eines Elements im Tupel ermitteln
# index("xy")
print("37 hat in t_gesamt den Index", t_gesamt.index(37))
```

```
37 hat in l_gesamt den Index 1
```

## Aufgabe

1. Es ist ein Tupel mit 5 Obstsorten (Datentyp str) zu erstellen und auf der Konsole auszugeben

2. Zwei weitere Tupel sind zu erstellen, das eine enthält 4 Milchprodukte und einem leeren String, das andere 2 Backwaren. Beide Tupel sind auf der Konsole nacheinander auszugeben.

3. Alle 3 Tupel sind in einem neuen Tupel zu verketten. Auf der Konsole ist das Tupel selbst, sowie der Datentyp des Tupels (Hinweis: type() ) auszugeben. Weiterhin sind das erste und das letzte Element des Tupels sowie deren Typ auszugeben

4. Folgende Aufgabe wurde bereits gelöst:
   *Für eine Nachricht soll von einem Programm automatisch die Anrede formuliert werden. Von der Konsole sind Eingaben für Name, Geschlecht und die Uhrzeit als Stundenangabe einzulesen • Die Anrede soll je nach Tageszeit mit „Guten Morgen“ (0–9 Uhr), „Guten Tag“ (10–17), „Guten Abend (18–0 Uhr) beginnen und anschließend mit „Herr xxx“ bzw. „Frau xxx“ fortgesetzt werden. Für xxx soll der entsprechende Name eingesetzt werden*
   Das Programm ist so anzupassen, dass die Überprüfung auf gültige Eingaben (wie z.B: m, w, d, M, Mann, f, Frau, …) mit Hilfe eines Tupels der zugelassen Werte durchgeführt wird. Ebenfalls sollen die Begrüßungsformeln (Guten Morgen, Guten Tag und Guten Abend) in einem Tupel definiert werden und die Anrede mit entsprechenden Zugriffen daraus zugesetzt werden.