Python-Besonderheiten - ganz kurz

```
>>> x=5
>>> x==5
True
>>> x<>5 #geht,obsolet
False
```

Wertzuweisungen und Vergleich

Das einfache Gleichheitszeichen ist eine Wertzuweisung, das doppelte dient zum Vergleichen. += geht, != statt <>.

Programmblöcke

```
def fak(n):
    if n==0: return 1
    return n*fak(n-1)
```

Programmblöcke beginnen mit einem Doppelpunkt und die Ebenen werden durch Einrückungen gekennzeichnet.

Datentypen

```
>>> 12**20
383375999244747512217 6L
>>> long(2e14)
200000000000000L
```

Langzahlarithmetik: Der Datentyp long ermöglicht das Rechnen mit praktisch beliebig langen ganzzahligen Werten. Python wechselt automatisch von integer zu long. Man kann float-Zahlen mit int und long umwandeln.

Strings

```
>>> s="wort"
>>> s
'wort'
>>> s=s+"e"
>>> s
'worte'
>>> s[1]
'o'
```

Stringkonstante kann man entweder mit doppelten Anführungszeichen oder mit einfachen definieren.
Strings sind statisch. Will man einen String bearbeiten, muss man ihn neu zuweisen.
Lesende Zugriffe auf einzelne Zeichen in Strings sind zulässig und funktionieren wie bei Listen (s.u.).

Zeichen

```
>>> zeichen="B"
>>> ord(zeichen)
66
>>> code=100
>>> chr(code)
'd'
```

Python kennt keinen Datentyp für Zeichen (character). Ein Zeichen ist ein String der Länge 1.
Die Anwendung von ord(...) für einen längeren String und die Anwendung von chr(...) für eine Zahl>255 führt zu einem Fehler.

Wahrheitswerte (bool)

Die beiden Konstanten für die Wahrheitswerte werden groß geschrieben: **`True`** und **`False`**.

Sequenzielle Datentypen

Listen

```
>>> tiere=['Hund','Katze','Maus']
>>> 'Maus' in tiere
True
>>> tiere+['Fliege']
['Hund', 'Katze', 'Maus', 'Fliege']
>>> tiere[1]
'Katze'
>>> tiere[1:2]
['Katze']
>>> tiere[:1]
['Hund']
>>> tiere[1:]
['Katze', 'Maus']
>>> len(tiere)
3
```

Listen […, ...] sind einer der Sequenziellen Datentypen, auf die über den Index (0-basierend) zugegriffen werden kann.
Viele Möglichkeiten bieten die Zugriffe auf ganze Abschnitte (slices).
[m:n] bedeutet "von m bis (vor) n".

Es geht auch "vom Anfang bis (vor) n" [:n] und "ab m bis zum Ende" [m:].
Listen sind veränderbar. Auch dabei kann man die Möglichkeiten von slices nutzen.
Die Inhalte von Listen sind nicht an bestimmte Typen gebunden.

Listen sind Objekte

Listen sind Objekte, die bei Zuweisung zu einem neuen Variablennamen nicht zum Erzeugen eines neuen Objektes führen! Das hat zur Folge, dass der Wert beider Variablen geändert wird.

```
>>> tiere2=tiere
>>> tiere2 += ['Pute']
>>> tiere
['Hund', 'Katze', 'Maus', 'Pute']
```

Tupel

Tupel (1, 2, 3) sind ähnlich den Listen, sind aber statisch.
Die lesenden Zugriffe sind also wie bei den Listen zulässig.

Nicht mehr sequenziell

Dictionary

Ein Dictionary hält das, was der Name verspricht.
Ein Dictionary-Objekt ist nicht statisch und bietet viele Zugriffsmöglichkeiten, z.B. über die Methode has_key.
Die Reihenfolge der Elemente ist nicht definiert.

```
>>> d={"eins" : 1, "zwei" : 2, "drei" : 3}
>>> d["zwei"]
2
>>> d.update({'vier' : 4})
>>> d
{'drei': 3, 'vier': 4, 'eins': 1, 'zwei': 2}
```

Mengen

Sets sind ebenfalls nutzbar. Hier ist gewährleistet, dass ein Element maximal einmal enthalten sein kann.
Die set-Objekte bieten die nowendigen Methoden wie z.B. Vereinigung (union) usw.

```
>>> s=set(['Hund','Katze'])
>>> len(s)
2
>>> s.add('Hund')
>>> len(s)
2
```

Verzweigungen

Verzweigungen kann man mit if … elif … else realisieren.

Globale Variable

Das Beispiel zeigt gleichzeitig das Verwenden einer globalen Variablen. Sie wird in der darauf zugreifenden Funktion als global gekennzeichnet. Wird derselbe Variablenname in einer anderen Funktion verwendet, in der diese Variable nicht als global gekennzeichnet ist, ist sie dort nur lokal und nicht mit der globalen identisch. Zuweisungen auf sie haben auf die globale keinen Einfluss.

```
stand= 'rot'
def schalte():
    global stand
    if stand == 'rot':
        stand = 'rot-gelb'
    elif stand == 'rot-gelb':
        stand = 'gruen'
    elif stand == 'gruen':
        stand = 'gelb'
    else:
        stand = 'rot'
```

Vordefinierte Parameter

```
def differenz(v1, v2=(0,0,0)):
    return ((v1[0]-v2[0]),(v1[1]-v2[1]),(v1[2]-v2[2]))

print differenz((6,4,2), (1,2,3))
print differenz((1,2,3))
```

Bei der Definition von Funktionen (Prozeduren, Methoden) können einzelne Parameter vordefinierte Werte bekommen. Bekommt die Funktion dann auch für den Parameter einen Wert übergeben, ersetzt er den vorgegebenen Wert.

Wiederholungen

Wiederholungen (Schleifen) kann man mit **`for`** und **`while`** realisieren.
Bei **`for`** folgt hinter **`in`** die Sammlung, über die iteriert werden soll oder der Iterator.
`range` erzeugt einen Iterator und damit werden die Elemente erst generiert, wenn sie benötigt werden.
`range(n)` erzeugt einen Iterator für Zahlen von 0 bis n-1.
`range(3,7)` geht auch und ergibt die Möglichkeit über [3,4,5,6] zu iterieren.
`range(3,10,2)` ergibt die Möglichkeit über [3,5,7,9] zu iterieren.

```
def ungerade(n):
    ergebnis=[]
    for i in xrange(n):
        ergebnis.append(2*i+1)
    return ergebnis

def gerade(n):
    i=0
    ergebnis=[]
    while (i < n):
        ergebnis.append(2*i)
        i += 1
    return ergebnis
```