



Python bietet folgende Standard-Datentypen:

Zahl

[Text ist geringfügig bearbeitet]

Zahlen gibt es als

- Integer-Zahl
- lange Integer-Zahl mit unbegrenzter (!) Größe
- Gleitkommazahl

Alle Zahlen sind miteinander kombinierbar, bei verschiedenen Typen wird automatisch zum mächtigeren (umfangreicheren) hin konvertiert.

```
intzahl = 5
lange_intzahl = 5L
gleitkommazahl = 5.0
```

Diese Zeilen nacheinander in die Python „shell“ eingeben

```
print 'Zahl: %d, Typ %s' % (intzahl, type(intzahl))
print 'Zahl: %ld, Typ %s' % (lange_intzahl, type(lange_intzahl))
print 'Zahl: %0.5f, Typ %s' % (gleitkommazahl, type(gleitkommazahl))
print
```

Hier wurden Zeichenkettenrepräsentationen in der bei "printf" üblichen Form in eine Zeichenkette "hineingewebt". Hinter der Zeichenkette folgt nach einem Prozentzeichen das Objekt, dessen Textdarstellung eingefügt werden soll. Sollen mehrere Objekte eingefügt werden, so muss ein Tupel (Zahlenpaar, Zahlentripel, ...) auf das Prozentzeichen folgen. Alles, was nicht "Zahl" ist, wird als String ("%s") notiert.

Text

Texte sind in der Länge nicht begrenzt. Textkonstanten werden in einfachen Hochkommata oder doppelten Anführungszeichen eingeschlossen notiert. Am Ende muss dann dasselbe Zeichen stehen.

```
txt1 = 'Ein Text'
txt2 = "Noch ein Text"
txt3 = 'Das Zeichen "\101" kann auch als "\\101" in Oktalschreibweise1 notiert werden'
txt4 = "Es gibt die 'üblichen' Ersatzdarstellungen:\t für Tabulator z.B."
print txt1
print txt2
print txt3
print txt4
print
```

Texte können nicht geändert werden, man kann jedoch sehr leicht aus einem bestehenden Text einen veränderten erzeugen. Es gibt ein String-Modul², das unter anderem folgende Dienste bietet:

- Klein- und Großbuchstabenwandlung, Übersetzungen
- Suchen, Ersetzen, Zerlegen, Zusammenfügen, Leerzeichen entfernen usw.

1 Oktalschreibweise bedeutet, dass die Ziffern zum Achtersystem gehören, also „normal“ geschrieben werden von der 0 bis zur 7, dann aber 8 → \10 , 9 → \11 , 10 (also „zehn“) → \12 (acht und zwei, also immer noch zehn, usw.)

2 Der Modul oder das ... - bei mir heißt es das.

Boolean

Einen besonderen Typ für Wahrheitswerte kennt Python *nicht*! Jedes Objekt kann als Bedingung aufgefasst werden; einige besondere Objekte, nämlich alle, die leer oder Null sind, sowie "None" gelten als "falsch", alle anderen als "wahr".

```
objekte = [0, 1, 0.0, 0.1, 0L, 1L, (), ('a',), [], [0], '', 'Emil', 'None', \
           None, {}, {1:'a'}]

for o in objekte:
    print 'Das Objekt', o, 'vom Typ', type(o), 'ist',
    if o:
        print 'wahr'
    else:
        print 'falsch'
```

Ausgabe s.u.

Als Ergebnis eines Vergleiches werden 0 bzw. 1 zurückgeliefert.

Ausgaben:

```
Das Objekt 0 vom Typ <type 'int'> ist falsch
Das Objekt 1 vom Typ <type 'int'> ist wahr
Das Objekt 0.0 vom Typ <type 'float'> ist falsch
Das Objekt 0.1 vom Typ <type 'float'> ist wahr
Das Objekt 0 vom Typ <type 'long'> ist falsch
Das Objekt 1 vom Typ <type 'long'> ist wahr
Das Objekt () vom Typ <type 'tuple'> ist falsch
Das Objekt ('a',) vom Typ <type 'tuple'> ist wahr
Das Objekt [] vom Typ <type 'list'> ist falsch
Das Objekt [0] vom Typ <type 'list'> ist wahr
Das Objekt vom Typ <type 'str'> ist falsch
Das Objekt Emil vom Typ <type 'str'> ist wahr
Das Objekt None vom Typ <type 'str'> ist wahr
Das Objekt None vom Typ <type 'NoneType'> ist falsch
Das Objekt {} vom Typ <type 'dict'> ist falsch
Das Objekt {1: 'a'} vom Typ <type 'dict'> ist wahr
```

Selbstdefinierte Datentypen

gibt es nicht; man kann den gewünschten Effekt jedoch mit Klassen erreichen.