

## Variablen / Wertzuweisungen

Variablen müssen nicht deklariert werden, der Typ (beispielsweise int, float, string) wird automatisch bei Zuweisung festgelegt.

Beispiel:

```
>>> zahl=1
>>> preis=5.98
>>> name="Thomas"
```

Die Zeichen ">>>" sind der Befehlsprompt des Interpreters bei interaktiver Eingabe.

## Anweisungsfolgen

Im Python wird immer genau eine Anweisung pro Zeile notiert. Daher ist es nicht nötig, mit Semikolons am Zeilenende zu arbeiten. Es existieren auch keine Blockmarkierungen wie etwa "{" oder "begin" "end". Stattdessen werden Blöcke an der Einrückung des Quelltextes erkannt. Python Quelltext erzwingt dadurch eine saubere Notation und gilt daher als besonders gut lesbar.

Zur Notation einzelner Kommentare wird Zeichen "#" verwendet:

```
# Kommentar
```

Mehrzeilige Kommentare können in dreifachen Anführungszeichen notiert werden:

```
"""
    langer Kommentar
"""
```

## Ein- und Ausgabe

print gibt Texte und Variablen aus. Einzelne Werte können durch Kommata getrennt werden.

```
print "Hallo Welt"
print "Preis : ", preis
```

Die Funktionen input() bzw. raw\_input() geben eine Meldung aus und warten auf Eingabe von einer Zahl oder einem String:

```
wert = input("Bitte einen Wert eingeben." )
name = raw_input("Bitte den Namen eingeben." )
```

Zur Notation von Strings können statt Anführungszeichen auch einfache Hochkommata verwendet werden.

```
print 'Ich sage "Hallo". '
```

## Operatoren

(+, -, *, /)	für Addition, Subtraktion, Multiplikation, Division
==	für Vergleiche auf Gleichheit
!= oder <>	für Vergleiche auf Ungleich
<, <=, >, >=	für weitere Vergleiche
not	ausdruck für ein logisches Nicht
or, and	logisches oder bzw. und

```
>>> print 2*(3+4)
14
```

+ und \* ist auf sehr viele Typen definiert, beispielsweise auch auf Strings und Listen.

```
>>> print "Hallo" + " Welt"
Hallo Welt
>>> print 3 * "Hallo!"
Hallo!Hallo!Hallo!
```

## wichtige Funktionen

text=str(wert)	Wandelt einen Wert zu einem String
wert=eval(str)	Wandelt einen Text zu einem Wert. Der Text darf auch eine Funktion sein und auch Variablen enthalten!

## Fallunterscheidungen

... werden durch if eingeleitet, weitere Fälle werden mit elif und else behandelt. Die zugehörigen Anweisungsblöcke müssen eingerückt werden !!!

```
if x<0:
    print "x ist negativ"
elif x==0:
    print "x ist Null"
else:
    print "x ist positiv"
```

Achtung: zum Vergleich auf Gleichheit wird == verwendet.

Blockbildende Anweisungen wie Fallunterscheidungen, Schleifen oder Funktionsdefinitionen werden immer mit einem Doppelpunkt notiert.

## Schleifen

Python kennt Wiederholungsschleifen (while) und Iterationsschleifen (for). Zum vorzeitigen Verlassen einer Schleife wird break verwendet. Die zugehörigen Anweisungsblöcke müssen eingerückt werden !!!

```
x=1:
while x<10:
    x=x+1
```

Es gibt keine Repeat-Schleife, statt dessen wird mit if ... break gearbeitet:

```
x=1
while 1:
    x=x+1
    if x>10: break
```

## erweiterte Datentypen, Listen

Es gibt Listen, Tupel (Wertpaare) und Hashes (Dictionaries), welche direkt in die Syntax integriert sind. Dadurch sind diese häufig verwendeten Datenstrukturen besonders gut lesbar und benötigen keine separaten Module. Hier kurz zu den Listen:

Liste erzeugen:

```
>>> liste = [0,1,2,3,4]
```

Ein Element auslesen:

```
>>> print liste[2]
2
```

Die Indizierung in Listen beginnt bei 0 und ist immer exklusiv des letzten Elements.

Einen Bereich auslesen:

```
>>> print liste[1:4]
[1, 2, 3]
```

Durch Weglassen einer Intervallgrenze erhält man den Anfang oder das Ende der Liste:

```
>>> print [:3]
[0,1,2]
```

Methoden auf Listen:

<code>liste.append(5)</code>	Ein Element einfügen:
<code>liste.count(3)</code>	Wie oft gibt es das Element?
<code>liste.remove(3)</code>	Lösche dieses Element.
<code>liste.index(3)</code>	An welcher Stelle ist das Element?
<code>liste.sort()</code>	Sortiere die Liste

### For-Schleifen

... arbeiten typischerweise nicht auf Intervallen (wie Pascal), sondern auf Listen oder Objektmengen:

```
for wert in liste:
    print wert
```

Werte-Intervalle können ggf. mit der Funktion `range(von, bis)` erzeugt werden:

```
for wert in range(1,10):
    print wert
```

### Module

Zum Importieren von kompletten Modulen wird die Anweisung `from modulname import *` verwendet.

Wichtige Module:

<code>math</code>	Mathematische Funktionen wie <code>pi</code> , <code>sin()</code> , <code>cos()</code>
<code>random</code>	Zufallszahlen: <code>x=randrange(2,20)</code>
<code>sum</code>	Schulmodul Stifte und Mäuse

### Funktionen

Funktionen werden mit `def` gefolgt von Name und Parameterliste definiert und können einen Wert mit `return` zurückgeben. Alle Variablen und Parameter in der Funktion gelten lokal, das heißt sie werden nach Verlassen der Funktion gelöscht. Funktionen sind Polymorph, das heißt der Typ der Parameter muss nicht festgelegt werden.

```
>>> def addiere(x,y):
>>>     summy=x+y
>>>     return summe
>>> print addiere(3,5)
8
```

### Ausnahmebehandlung

Zur Erkennung und Behandlung von Fehlern beim Programmablauf kann die so genannte Ausnahmebehandlung verwendet werden:

```
# Sichere Division
try:
    print "x durch y ist:" , x/y
except:
    print "Division nicht durchführbar."
```

### Klassen und Objekte

An diese Stelle direkt ein Beispiel. Bei der Definition Methoden (aber nicht beim Aufruf) muss `self` immer als erster Parameter angegeben werden. Die Methode `__init__` ist der Konstruktor, er wird bei Erzeugung einer Objektinstanz automatisch aufgerufen.

```
class Person:
    def __init__(self, name):
        # Dies ist der Konstruktor,
        # hier werden auch Attribute
        # festgelegt.
        self.name=name
        self.alter=1

    def setzeDaten(self, alter):
        # Diese Methode ändert den Namen
        self.alter= alter

    def liesAlter(self):
        # Diese Methode liefert ein Attribut
        return self.alter

>>> p=Person("Thomas")
>>> p.setzeAlter(16)
>>> print p.liesAlter()
16
```

Attribute und Methoden einer Klasse sind immer öffentlich. Python bietet eine automatische Speicherverwaltung, es ist daher nicht nötig, Speicher zu reservieren oder freizugeben.

### Vererbung

Soll eine Klasse von einer anderen erben, wird bei Definition der neuen Klasse die Elternklasse in Klammern angegeben.

Beispiel:

```
class Schueler(Person):
    def setzeSchulklasse(self, klasse):
        self.klasse=klasse
```

### Python-Quellen

Python	<a href="http://www.python.org">www.python.org</a>
Stifte und Mäuse für Python, PyNassi	<a href="http://www.ingo-linkweiler.de/diplom">www.ingo-linkweiler.de/diplom</a>

März 2003, erstellt von: Ingo Linkweiler, [i.linkweiler@gmx.de](mailto:i.linkweiler@gmx.de)