

Python Kurzreferenz

www.python.org (aktuell ist Version 3, wir verwenden noch die Version 2.7)

nach der Installation finden wir ein Programm namens 'Idle' im Python-Startmenü.

Der Python-Prompt lautet >>> und bedeutet, dass Python bereit ist, eine Anweisung auszuführen oder eine Frage zu beantworten.

Python arbeitet interaktiv – wir können eine Anweisung schreiben und sofort die Auswertung erhalten. Python antwortet also mit der ausgewerteten Eingabe.

```
>>> print "Hallo Welt!"
Hallo Welt!
>>> "Hallo Welt"
'Hallo Welt!'
```

Die Grundrechenarten

>>> 5-3+2-7 -3	Addition und Subtraktion
>>> 4-(3-1)-(1-(2-3)) 0	Klammern
>>> 3*5 15	Multiplikation
>>> 1+3*5+1 17	Vorrangregeln
>>> 1.2*3 3.6	Dezimalzahl (floating point)
>>> 7/4 1	Division (Version 3 volle Division) Version 2: wie oft geht 4 in 7 ?
>>> 7%4 3	Divisionsrest
>>> 7//4 1	Ganzzahldivision
>>> 7.0//4.0 1.0	
>>> 7.0/4.0 1.75	floating point Division
>>> 2**10 1024	Exponentiation
>>> 2**100 1267650600228229401496703205376L	eine große Zahl L bedeutet 'long integer' (nur Version 2)
>>> print(2**100) 1267650600228229401496703205376	
>>> 1.0/7 0.14285714285714285	Ergebnis mit Genauigkeit der CPU
>>> print 1.0/7 0.142857142857	gerundet auf 12 Stellen (Standard)
>>> print "%3.7f"%(100.0/7) 14.2857143	Rundung 3 Vorkomma, 7 Nachkommastellen

Wichtige Formatierungszeichen: \ ' \" Anführungszeichen, \n linefeed, \t tab

Stringformatierung %[flags][width][.precision]typecode

flags: - linksbündig, + Vorzeichen, <spc> Leerzeichen vor positiv, 0 mit Nullen füllen

width: gesamte Feldbreite

precision: Anzahl der Nachkommastellen

typecodes:

%s String, %c Character (int oder str), %d decimal, %x %X hex, %b binary

%f %F floating point %e %E floating point mit Exponent %% Prozentzeichen

Variable

Bezeichner dürfen Buchstaben und Ziffern, sowie den underscore enthalten und nicht mit einer Ziffer beginnen. In Python bevorzugt man Kleinschreibung (Unterschied zu camelcase in Java)

```
>>> x = 3
>>> zehn = 10
>>> wert_23 = 11
>>> print x*zehn+wert_23
41
```

Wertzuweisungen

```
>>> x = y = x2 = 3
>>> x,y = 10,20
>>> x,y = y,x
>>> print x,y
20,10
```

macht x und y und x2 zu 3
Mehrfachzuweisung. x = 10 und y = 20
rechts auswerten, dann links zuweisen

Die Werte wurden getauscht

Strings

```
>>> a,b = "Hallo","Du!"
>>> print a+" "+b
'Hallo Du!'
>>> print b*3
Du!Du!Du!
>>> len(a)
5
```

zwei Zeichenketten
Stringaddition

Stringmultiplikation

die Länge des Strings
(= Anzahl der Buchstaben)

Typ

```
>>> x,y,z = 2,3.1,"23"
>>> type(x)
<type 'int'>
>>> type(y)
<type 'float'>
>>> type(z)
<type 'str'>
```

die Funktion **type** liefert den Typ

Elemente und Slices (Schnitte)

Indizes beginnen immer mit Null.

Bei Bereichen gilt der erste Index (Startwert) inklusive, der zweite (Abbruchwert) exklusive.

```
>>> s = "abcdefgh"
>>> s[2]
'c'
>>> s[-1]
'h'
>>> s[2:5]
'cde'
>>> s[1:-1]
'bcdefg'
>>> s[:3]
'abc'
>>> s[6:]
'gh'
```

ein Element bei Index 2

ein Element von hinten

Teilstring von Index 2 bis vor 5

von vorne

bis ans Ende

<pre>>>> s[1:7:2] 'bdf' >>> s[8:1:-1] 'hgfedc'</pre>	<p>von 1 bis vor 7 in Zweierschritten</p>
--	---

Wahrheitswerte und logische Ausdrücke (boolean expressions)

alle Werte, die Null oder leer sind gelten als False, die übrigen als True

<pre>>>> 3 == 2+1 True >>> 5 != 10/2 False >>> not 5==4 True >>> 5 > 3 True >>> -10 <= 1 True >>> (3>1) and (2<10) True >>> (3>1) or (2>=10) True >>> "x" in "uvWxY" True >>>"x" not in "uvWXY" True >>> x = [2,3,4] >>> y = x >>> x is y True >>> y = [2,3,4] oder y = x[:] >>> x is y False >>> x==y True</pre>	<p>gleicher Wert</p> <p>ungleich</p> <p>Negation</p> <p>beide Ausdrücke sind wahr</p> <p>mindestens einer der Werte ist wahr</p> <p>Element ist enthalten</p> <p>Element ist nicht enthalten</p> <p>gleiches Objekt die gleiche Liste, zwei Zeiger darauf</p> <p>eine neue Liste mit gleichem Inhalt</p> <p>unterschiedliche Zeiger</p> <p>aber gleiche Werte</p>
---	---

Wiederholungen

<pre>>>> for x in "abc": print x a b c >>> for x in [4,5,-6]: print x**2 16 25 36</pre>	<p>for-Schleife. Die Laufvariable x nimmt nacheinander jeden Wert an</p> <p>jeder Wert der Liste wird angenommen</p>
---	--

Zahlenbereiche

<pre>>>> range(8) 0,1,2,3,4,5,6,7 >>> range(3,6) [3,4,5]</pre>	<p>die ersten 8 natürlichen Zahlen ab Null</p> <p>die ganzen Zahlen von inkl. 3 bis excl. 6</p>
--	---

<pre>>>> range(-3,11,2) [-3,-1,1,3,5,7,9] >>> range(6,1) [] >>> range(6,1,-1) [6,5,4,3,2]</pre>	<p>ganze Zahlen von -3 bis vor 11 in Zweierschritten</p> <p>6 ist schon größer als der Endwert abwärts zählen</p>
--	---

eigene Funktionen definieren

<pre>>>> def hallo(s): print "Hallo, lieber "+s >>> hallo("Lehrer") 'hallo lieber Lehrer' >>> def flaeche(a): f = a*a return f >>> flaeche(2.5) 6.25</pre>	<p>gibt einen neuen String aus</p> <p>ergibt die Fläche eines Quadrats</p> <p>Rückgabewert Funktionsaufruf</p>
---	--

Man kann zwei Arten von Funktionen unterscheiden: die einen tun nur etwas und liefern keinen Wert zurück, die anderen tun etwas und liefern einen Wert (oder mehrere Werte) an den Aufrufer zurück, die von diesem weiter bearbeitet werden können.

Entscheidungen treffen

<pre>>>> if 10>2: print "stimmt" 'stimmt' >>> if x<10: print "kleiner" else: print "nicht kleiner" >>> if x<0: print "kleiner" elif x>0: print "größer" else: print "gleich" if x<0: print "kleiner" elif x>0: print "größer" else: print "gleich"</pre>	<p>ist der auszuführende Block nur eine einzige Zeile, muss er nicht in einer neuen Zeile stehen</p>
--	--

weitere Wiederholungsanweisungen

<pre>while Bedingung: break continue</pre>	<p>während die Bedingung gilt, mach : . .</p> <p>verlässt diese Schleife sofort und macht nach der Schleife weiter</p> <p>beendet diesen Schleifendurchlauf und macht sofort mit dem nächsten Durchlauf weiter</p>
--	--

print – Python 2

in dieser Version ist print eine ganz spezielle Anweisung, die die nachfolgenden Werte als Textzeichen zum Bildschirm schickt. Sie braucht keine Klammern um ihre Argumente.

<pre>>>> print "hallo",1,2,3 hallo 1 2 3 >>>print 1,2, ; print 3,4 1 2 3 4</pre>	<p>Argumente werden mit Beistrichen getrennt und durch Leerzeichen getrennt angezeigt</p> <p>ein nachfolgendes Komma unterdrückt den Zeilenvorschub in der Ausgabe</p>
---	--

Testmöglichkeit in der IDLE: man darf mehrere Anweisungen durch Strichpunkte getrennt in eine einzige Zeile schreiben.

print – Python 3

hier ist print eine ganz gewöhnliche Funktion (die Parameter stehen deshalb immer in Klammer), die in der Standardeinstellung Werte als Textzeichen zum Bildschirm schickt. Sie besitzt zusätzliche Argumente:

sep="..." das Trennzeichen zwischen den einzelnen Werten, default ' '

end=" " das Zeilenendzeichen, default '\n'

Vorteile: print benötigt keine 'Sondereigenschaften' mehr und hat noch weitere praktische Fähigkeiten.

<pre>>>> print("hallo",1,2,3) hallo 1 2 3 >>>print(1,2,7,sep='!',end='~'); print(3,4) 1!2!7~3 4</pre>	<p>Argumente werden mit Beistrichen getrennt und durch Leerzeichen getrennt angezeigt</p> <p>ein nachfolgendes Komma unterdrückt den Zeilenvorschub in der Ausgabe</p>
--	--

Was ist bei der Umänderung auf Version 3 zu tun?

- die zu druckenden Werte in Klammer setzen
- bei unterdrücktem Zeilenvorschub sep=' ' hinzufügen

Schreibt man in Version 2 immer mit formatierten prints wie "%d"%wert kann man die Klammer ruhig setzen und braucht nichts zu adaptieren.

Eingabe - Python 2

Tastatureingaben werden wie folgt eingelesen:

<pre>>>> x = raw_input("Bite eine Zahl: ") >>> zahl = int(x)</pre>	<p>Die eingelesene Zeile ist als String x verfügbar</p> <p>dieser String kann in eine Zahl umgewandelt werden</p>
--	---

Python 2 kennt auch eine Funktion namens input: sie wandelt den eingelesenen String sofort in seinen Wert um (sie wertet den String aus).

```
input = eval(raw_input())
```

Wegen dieser einfachen Möglichkeit (die außerdem noch die Behandlung von Eingabefehlern ermöglicht) ist der Befehl input eigentlich überflüssig.

Übrigens kann man in Python beliebige Ausdrücke eingeben, die mittels eval() auswertbar sind

<pre>>>> x = raw_input("frag mich was: ") >>> eval(x) 19</pre>	<p>ich gebe 3+2**4 ein</p>
--	----------------------------

Eingabe - Python 3

raw_input heißt nun einfach input, raw_input gibt es nicht mehr

>>> x = input("Bitte eine Zahl: ")	Die eingelesenen Zeichen sind als String x verfügbar
------------------------------------	--

Auswertung oder Umwandlung wie üblich mittels eval(), int(), float(),...
Grund der Umstellung: ohne Fehlerprüfung ist das alte input() sehr fehlerträchtig und ein Sicherheitsrisiko.

Bibliotheken

Für spezielle Aufgaben sind in Python zahlreiche Bibliotheken vorhanden, die man nur dann lädt, wenn man sie tatsächlich benötigt. Das spart Speicherplatz.

Als Beispiel möchte ich die Funktion ctime() in der Bibliothek time aufrufen, die mir das aktuelle Datum mit der Uhrzeit angibt

>>> import time >>> time.ctime() 'Thu Nov 03 10:35:13 2011'	Bibliothek einbinden Funktionen der Bibliothek aufrufen
>>> from time import ctime >>> ctime()	eine Einzelfunktion laden, als wäre sie lokal in meinem Programm
>>> from time import *	holt alle verfügbaren Funktionen lokal ins Programm

Beachte den Unterschied im Aufruf:

- import modul

hier bleibt das zugefügte Modul ein externes Paket, auf dessen Inhalt wir durch den vorangestellten Modulnamen (und einen trennenden Punkt) zugreifen können.

- from modul import

hier werden alle im Modul enthaltenen Bestandteile direkt in unser Programm integriert. Der getrennte Namensraum geht verloren, wir müssen darauf achten, nicht mit Bezeichnern aus dem Modul zu kollidieren. Wenn man weiß, was man tut, spart diese Methode aber oft viel Tipparbeit. Speziell Grafikbibliotheken werden meist so eingebunden.

>>> import math >>> math.pi 3.141592653589793 >>> pi = 3 >>> print math.pi, pi 3.141592653589793 3	getrennter Namensraum das Original ist unbeschädigt
>>> from math import * >>> pi 3.141592653589793 >>> pi = 3 >>> print pi 3	der korrekte Wert ist verloren

einfache Standardtypen

123 123.45 3.14e+2	Zahlen
--------------------------	--------

<pre>4. .6 0o177 0x1ff 0b111 3.2+4j int() float() complex() hex() oct() bin() "hallo" 'hallo' "I'm here" 'say "yes!"' 'You\'re here' "automatisch" "zusammen" "gesetzt" """mehrzeiliger Text..."""</pre>	<pre>oktal hexagesimal binär komplex Umwandlungsfunktionen Darstellung (strings!) Strings multiline string</pre>
---	--

zusammengesetzte Standardtypen

<pre>[3, 'hi', 12]</pre>	list : Einträge änderbar
<pre>(3, 'hi', 12)</pre>	tupel : Einträge unveränderlich
<pre>{'ich':'super', 'du':'auch super'} {3 : 20, -10:1, 'hi':'hallo', 'A':2}</pre>	dictionary : Einträge veränderlich (hash)

Klassen, Methoden, Attribute

Python beherrscht die moderne objektorientierte Programmierung

<pre>class Meineklasse: x = 0 y = "Wort" def __init__(self, startwert): self.startwert = startwert self.x = self.x + startwert self.rechne(self.x*2) def rechne(self, wert): self.a = self.y * wert</pre>	<pre>definiert die Klasse Klassenvariable (Attribute) diese Methode wird beim Erzeugen ausgeführt lokaler Wert in ein Attribut gespeichert Zugriff auf Attribute Aufruf einer Klassenmethode eine Methode a wird berechnet und notfalls erzeugt</pre>
---	---

Nun ein Beispiel: wir modellieren ein Kartenspiel.

Seine Eigenschaften (Attribute):

die vorrätigen Spielkarten

Seine Fähigkeiten (Methoden):

die Spielkarten erzeugen `__init__()`

die Karten mischen `mische()`

Karten herzeigen `zeige(anzahl)`

Karten geben `gib(anzahl)`

```

import random

class Spielkarten():

    farben = ["Treff", "Pik", "Karo", "Herz"]
    werte = ["2", "3", "4", "5", "6", "7", "8", "9", "10", "Bub", "Dame", "König", "As"]
    karten = []

    def __init__(self):
        for f in self.farben:
            for w in self.werte:
                self.karten.append(f+"-"+w)

    def mische(self):
        random.shuffle(self.karten)

    def zeige(self, anzahl=-1):
        if anzahl<0: anzahl = len(self.karten)
        return self.karten[:anzahl]

    def gib(self, anzahl):
        if anzahl<1 or anzahl>len(self.karten):
            print "So viele Karten kann ich nicht geben!"
            return False
        gegeben, self.karten = self.karten[:anzahl], self.karten[anzahl:]
        return gegeben

```

nun testen wir in der Idle:

```

>>> spiel = Spielkarten()
>>> spiel.zeige(4)
['Treff-2', 'Treff-3', 'Treff-4', 'Treff-5']

>>> spiel.mische()
>>> spiel.zeige(4)
['Pik-Dame', 'Herz-4', 'Karo-Bub', 'Treff-9']

>>> spiel.gib(2)
['Pik-Dame', 'Herz-4']

>>> spiel.gib(3)
['Karo-Bub', 'Treff-9', 'Karo-3']

```

abschließende Tips

<pre> # -*- coding: iso-8859-1 -*- #!/usr/bin/python # -*- coding: UTF-8 -*- </pre>	<p>diese Zeile am Anfang Deiner .py-Datei definiert die länderspezifische Codepage. Damit werden Sonderzeichen auch außerhalb Deiner IDLE korrekt wiedergegeben.</p> <p>so könnte der Dateistart einer Web-Applikation auf einem Linux Webserver aussehen</p> <p>'shebang' und Programmpfad codepage für Ausgaben</p>
--	---

<pre> from __future__ import print_function </pre>	<p>diese Zeile erlaubt es Dir, bereits in Python 2.7 die neue print-Syntax von Python 3.0 zu verwenden.</p>
--	---

<code>from __future__ import division</code>	erzwingt die neue Version der Division: nur <code>//</code> ist die Ganzzahldivision
--	---

<pre>def eingabe(prompt): return input(prompt) def eingabe_raw(prompt): return raw_input(prompt) ----- def eingabe(prompt): return eval(input(prompt)) def eingabe_raw(prompt): return input(prompt)</pre>	<p>statt <code>input</code> zu verwenden in Py 2.7</p> <p>statt <code>raw_input</code> zu verwenden in Py 2.7</p> <p>für Python 3: nur zwei kleine Änderungen nötig</p>
--	---