Python Workshop

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Hinweis

Dieses Material dient der Ergänzung zum Python Workshop und ist keine Dokumentation der Funktionen von Python. Viel mehr ist es eine Sammlung von Beispiel, die im Workshop gezeigt wurden und zuhause noch einmal nachvollzogen werden können.

2 Python interaktiv

- Python bietet Kommandozeilen-Interpreter
- gut geeignet zum Testen
- kann auch als Taschenrechner verwendet werden

2.1 python

- standard Python-Interpreter
- steht immer zur Verfügung, wenn Python installiert ist
- Hilfe mit:
  - help() - allgemeine Python Hilfe
  - help(obj) - Hilfe zu einem Objekt, einer Funktion oder einem Modul
  - dir() - alle belegten Namen
  - dir(obj) - alle Attribute eines Objekts

```
$ python
Python 2.6.4rc2 (r264rc2:75497, Oct 20 2009, 02:55:11)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2.2 ipython

- erweiterter Python-Interpreter
- nutzt Python → kein eigener Interpreter
- bietet Nutzer mehr Hilfe
  - z.B.: ? Hinter-Objekt gibt Hilfe aus

```
$ ipython
Python 2.6.4rc2 (r264rc2:75497, Oct 20 2009, 02:55:11)
Type "copyright", "credits" or "license" for more information.

IPython 0.10 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object'. ?object also works, ?? prints more.

In [1]:
```

3 Zahlen in Python

3.1 Ganzzahlen

- negative und positive Ganzzahlen
- im allgemeinen vom Typ Integer
- für sehr große Zahlen Long
- Umwandlung von Integer in Long erfolgt automatisch
- Beispiel: 5, -10, +11

3.2 Fließkommazahlen

- negative und positive Zahlen mit Komma
- wichtig ist, dass ein .(Punkt) und kein ,(Komma) verwendet wird
- Achtung!!!: für Finanzberechnungen gibt es extra Pakete mit genaueren Zahlen
- Beispiel: 1.0, -3.0, +1.98

3.3 Imaginärzahlen

- imaginäre Zahlen
- Beispiel: (1,0), (1,-1)

3.4 Bool

- Werte: True, False
- Werte die als False Interpretiert werden
  - None
  - False
  - 0 (numerisch)
  - leere Strings, Listen und Tupel
  - leere Dictionaries
  - leere Sets

```
>>> True
True
>>> False
False
```

4 Rechnen in Python

4.1 Addition und Subtraktion

```
>>> 2+3
5
>>> 5+7
12
>>> 9-20
-11
>>> 1.5+0.4                                    (1)
1.8999999999999999
>>> (1+0j)+(0+1j)
(1+1j)
```

(1) Achtung: 1.5+0.4 ergibt 1.8999.. wegen Ungenauigkeit der Fließkommazahlen, andere Programmiersprachen haben gleiches Problem

4.2 Multiplikation und Division

```
>>> 1.5/1
1.5
>>> 1.5/2
0.75
>>> 1.5/2
0.75
>>> (1+0j)/(1+1j)
(0.5-0.5j)
```

4.3 Hoch und Wurzel

```
1 >>> 3**3
2 27
3 >>> 27**(1/3)                                 (1)
4 1
5 >>> 27**(1/3.0)
6 3.0
```

(1) Achtung: 1/3 ergibt 1, deswegen muss mit 3.0 gerechnet werden

4.4 Rest

```
>>> 4 % 3
1
```

5 Variablen

- Name darf aus Unterstrich, Buchstaben und Zahlen bestehen
- erstes Zeichen darf keine Zahl sein

- Wertzuweisung mit =

```
>>> foo = 5
>>> foo
5
>>> foo = 1.5
>>> foo
1.5
```

6 Arrays

6.1 Listen

- append() - Wert anhängen
- insert() - Wert an Position einfügen
- pop() - letzten Wert zurückgeben und löschen
- remove() - Wert löschen

```
>>> a = []
>>> a.append(1)
>>> a
[1]
>>> a.append(2)
>>> a
[1, 2]
>>> a.pop()
2
>>> a
[1]
```

6.2 Tuple

- sehr einfache Listen
- sind nicht veränderbar

```
>>> a = (1,2)
>>> a
(1, 2)
>>> (b,c) = a
>>> b
1
>>> c
2
```

6.3 Dict

- assoziative Arrays
- Operationen

- clear() - löscht alle Einträge
- copy() - kopieren
- get() - Wert mit Key zurückgeben
- has_key() - ist Key mit gegebenem Namen vorhanden
- items() - Tupel aus (Key, Wert)
- keys() - alle Keys
- values() - alle Werte

```
>>> a = {'wert1':1, 'wert2': 'zwei'}
>>> a
{'wert2': 'zwei', 'wert1': 1}
>>> b = dict(wert1 = 1, wert2 = 'zwei')
>>> b
{'wert2': 'zwei', 'wert1': 1}
>>> b['wert2']
'zwei'
>>> b['wert2'] = 'test'
>>> b['wert2']
'test'
```

7 Zeichenketten/Strings

7.1 Einfach

```
>>> "test"
'test'
>>> "test\n"
'test\n'
>>> 'test\n'
'test\n'
```

7.2 Erweitert

```
>>> """Text
... ueber
... mehrere
... Zeilen
... """
'Text\nueber\nmehrere\nZeilen\n'
```

7.3 Unicode

- wichtig, wenn zum Beispiel Umlaute verwendet werden sollen
- kann in sehr viele andere Zeichenkodierungen umgewandelt werden (Bsp.: utf-8)

```
>>> u"Über sieben Brücken"
u'\xdcber sieben Br\xfccken'
```

7.4 Arbeiten mit Zeichenketten

```
>>> foo = "Test Nachricht"
>>> foo[0]
'T'
>>> foo[2:4]
'st'
>>> foo[2:]
'st Nachricht'
>>> foo[:4]
'Test'
>>> foo[:-4]
'Test Nachr'
>>> foo[-1:]
't'
>>> foo[-3:]
'cht'
```

7.5 Weitere Funktionen

- globale Funktionen
  - len() - Länge der Zeichenkette
- Funktioen des String/Unicode-Objekt
  - find() - Zeichenkette in String finden
  - join() - füge eine Liste zusammen und nutze die Zeichenkette als Trennzeichen
  - replace() - Ersetze Zeichen oder Zeichenkette mit einem anderen Zeichen oder einer Anderen Zeichenkette
  - split() - an einem Zeichen oder einer Zeichenkette trennen
  - strip() - Whitespaces entfernen

```
>>> len('abc')
3
>>> 'abc'.find('b')
1
```

8 Logische Ausdrücke

8.1 Vergleichen

- <, >, ==, ⇐, >=, != , not, in

```
>>> 3 < 5
True
>>> 3 <= 5
True
>>> 3 != 5
True
>>> 3 == 5
False
>>> not 3 == 5
True
```

8.2 Verknüpfungen

- and, or

```
>>> 1 == 2 and 3 < 5
False
>>> 1 == 2 or 3 < 5
True
```

9 Ein- und Ausgabe

9.1 Einfach

- input - Liest Eingabe und für gleichzeitig eval() Funktion aus
- print - gibt Werte aus
- raw_input - Liefert Eingabe als String

```
>>> a = input('Test: ')
Test: 1
>>> print
1
>>> a = raw_input("Test: ")
Test: Foo
>>> print a
Foo
```

9.2 Formatierte Ausgabe

- Integer dezimal: d, i
- Integer oktal: o
- Integer hexadezimal: x, X
- Float: f, F
- Float in Exponentialdarstellung: e, E, g, G
- Einzelnes Zeichen: c
- String: s
- Zeichen % wird als %% dargestellt

```
>>> "Zahl: %i" % 5
'Zahl: 5'
>>> "Name: %s, Zahl: %i" % ('Mustermann', 5)
'Name: Mustermann, Zahl: 5'
```

9.3 Dateien

- open() - öffnet eine Datei und gibt ein File-Objekt zurück
- File-Objekt Funktionen
  - close() - schließt die Datei
  - read() - Zeichen lesen
  - readline() - eine Zeile lesen
  - readlines() - mehrere Zeilen lesen
  - write() - schreiben
  - writelines() - mehrere Zeilen schreiben

```
>>> fp = open('/tmp/test', 'w')
>>> fp.write('abc')
>>> fp.close()
>>> fp = open('/tmp/test')
>>> fp.read()
'abc'
>>> fp.close()
```

10 Fallunterscheidung

```
>>> if 1 == 2:
...     print "falsch"
...
>>> if 1 == 2:
...     print "falsch"
... else:
...     print "richtig?"
...
richtig?
>>> if 1 == 2:
...     print "falsch"
... elif 1 == 1:
...     print "richtig"
...
richtig
```

11 Schleifen

- for - einfache for-Schleife
- while - while schleife
- break - bricht schleife ab, auch wenn es Bedingungen nicht zulassen
- continue - fahre mit nächstem Element fort auch wenn Schleife noch nicht am Ende ist
- range(), xrange() - kann verwendet werden um Zählschleifen zu programmieren
- pass - Leerlauf-Anweisung

```
>>> for i in range(1,3):
...     print i
...
1
2
```

12 Funktionen

```
>>> def add(a, b):
...     return a+b
...
>>> add(1,2)
3
```

```
>>> def add(a, b = 2):
...     return a+b
...
>>> add(1,4)
5
>>> add(1)
3
```

13 Klassen

```
>>> class A(object):
...     a = 5
...     def set_a(self, a):
...         self.a = a
...     def get_a(self):
...         return self.a
...
>>> foo = A()
>>> foo.get_a()
5
>>> foo.set_a(1)
>>> foo.get_a()
1
```

- in diesem Beispiel erbt die Klasse B von der Klasse A

```
>>> class B(A):
...     b = 3
...     def set_b(self, b):
...         self.b = b
...     def get_b(self):
...         return self.b
...
>>> foo2 = B()
>>> foo2.get_a()
5
>>> foo2.get_b()
3
>>> foo2.set_a(1)
```

```
>>> foo2.set_b(2)
>>> foo2.get_a()
1
>>> foo2.get_b()
2
```

14 Python-Skript

- Code-Blöcke werden durch Einrücken gekennzeichnet
- Einrücken mit Tab oder Leerzeichen
- Empfohlen: 4 Leerzeichen
- Kommentare mit # gekennzeichnet

[source}

```
#!/usr/bin/env python

print "Hallo Welt"
```

15 Module und Pakete

15.1 Standard Packages

- math - mathematische Funktionen und Konstanten
- os - Betriebssystem spezifische Funktionen
- sys - Funktionen für das System

15.2 Verwendung

```
>>> import math
>>> math.pi
3.1415926535897931
>>> from math import pi
>>> pi
3.1415926535897931
>>> from math import pi as LustigeZahl
>>> LustigeZahl
3.1415926535897931
>>> from math import *
>>> sin(pi)
1.2246063538223773e-16
```

15.3 Pakete

- mehrere Module können hierarchisch zu Paketen zusammengefasst werden

```
>>> import os
>>> os.path.join('tmp', 'test')
'tmp/test'
```

15.4 Eigene Module

- jedes Python-Programm kann als Modul importiert werden

15.5 Eigene Pakete

- Pakete sind einfache Unterordner
- in jedem Unterordner muss eine Datei mit dem Namen *init*.py befinden
  - diese Datei kann leer sein

16 Fehler und Ausnahmen

```
>>> try:
...     1/0
... except:
...     print "Fehler"
...
Fehler
```

```
>>> try:
...     1/0
... except IOError:
...     print "IO-Fehler"
... except Exception, msg:
...     print str(msg)
...
integer division or modulo by zero
```

- Exceptions können mit der Funktion raise geworfen werden

17 Links

- Python.org
- Dokumentation auf Python.org