Python

Unter Windows empfiehlt sich der Download von Python unter folgender URL, http://www.python.org/download/. Linux Distributionen wie z.B. Ubuntu liefern Python direkt in Ihrer Paketverwaltung mit:

```
$ aptitue install python
```

Ein Python Script sollte mit der Dateiendung „.py“ gespeichert werden. Unter Windows kann man ein Script durch Doppelklick aus dem Explorer ausführen. Schliesst sich das Fenster sofort wieder, kann man ein „sys.stdin.readline()“ am Ende einfügen.
Unter Linux ruft man das Script im Terminal auf:

```
$ python myscript.py
```

Steht in der ersten Zeile #!/usr/bin/python und hat man die Ausführungsrechte gesetzt kann man es direkt ausführen.

Allgemeine Syntax

- Python ist casesensitiv
- Kommentare von ’#’ bis Zeilenende
- Kommando endet am Zeilenende
  - mehrere Kommandos pro Zeile mit „;“ möglich
  - Kommando über mehrere Zeilen mit „\“ am Zeilenende

- Blockbildung durch Einrücktiefe

```
if x > 2:
   x += 3   # if-Block, da tiefer eingerückt
print x     # if-Block beendet, da weniger eingerückt
```

- TABs und Spaces dürfen nicht gemischt werden!

- Umlaute in Scripts
  - bei Verwendung nicht 7-bit Zeichen Encoding in erster/zweiter Zeile angeben mit

```
# -*- coding: utf-8 -*-
```

Datentypen

Zahlen

Es gibt die Datentypen Integer, Float sowie Dezimalzahlen beliebiger Genauigkeit.

```
i=1   # Zuweisung
i+=1  # i++ wie in C gibt es nicht
i*=2

i**2  # i "hoch" 2
```

http://docs.python.org/library/stdtypes.html#typesnumeric

Strings

Strings können in einfachen oder doppelten Anführungszeichen geschrieben werden. Mehrzeilige Strings sind durch dreifache Anführungszeichen möglich.

```
s1='Er sagte "Hallo" zu mir'

s2="er ist der 20'te"

s3="""Mehrere Zeilen
Zeile 2
Zeile 3
"""
```

Die wie in C gewohnte Formatierte Ausgabe mit „printf“ ist auch möglich.

```
a="Alpha"
g="Gamma"
b="Beta"
output="%s, %s und %s" % (a, g, b)    # Ausgabe: Alpha, Gamma und Beta

# Formatieren von Float:
f=0.123456
s="%06.2f" % f    # Ausgabe: 000.12
```

Listen

„Strings“ und „Tuples“ können nicht verändert werden. „Listen“ sind im Gegensatz dazu veränderbar. Es können beliebig Listeneinträge hinzugefügt oder entfernt werden.

```
list=["Das", "Wetter"]
list.append("schön")
list[1]="Leben"
list.insert(2, "ist")
print list             # Ergebnis: Das Leben ist schön
```

Listen von Listen:

```
list[1]=[3, 4]
```

http://docs.python.org/library/stdtypes.html#typesseq-mutable

Dictionaries (Hash-Tables)

Dictionaries bestehen aus einen Schlüssel (Key) und einem zugeordneten Wert (Value).

```
autos={}
autos["Klaus"]="BMW"
autos["Bernd"]="Lada Niva"
autos["Frank"]="Porsche"

print "Bernd: ", autos["Bernd"]
print "Was steht alles drin: ", autos

# Alle Einträge ausgeben:
for name, auto in autos.items():
      print "Name: %s, Auto: %s" % (name, auto)
```

Wichtige Funktionen:

Funktion	Rückgabewert	Beschreibung
`mydict.keys()`	`[key1, key2, ...]`	Alle Schlüssel
`mydict.values()`	`[value1, value2, ...]`	Alle Werte
`mydict.items()`	`[(key1, value1), (key2, value2), ...]`	Liste von 2er Tuples
`mydict.has_key(key)`	`True oder False`	Schlüssel vorhanden?
`mydict.get(key1, default)`	`value1 oder default`	Wert von „key1“ falls nicht vorhanden: „default“

Listen

List Comprehensions

List Comprehensions ist die Möglichkeit aus einer alten Liste eine neue zu erzeugen. List Comprehensions kann optional auch Bedingungen enthalten:

```
a = [2,1,3]
b = [2*x for x in a if x < 3]# b == [4,2]
```

Beispiel:

```
# Mit einer Schleife Leerzeichen am Anfang/Ende entfernen.
fruechte=[' Apfel  ', ' Banane ', '  Kiwi ', ' Ananas']
neue_fruechte=[]
for frucht in fruechte:
      frucht=frucht.strip()
      neue_fruechte.append(frucht.strip())

fruechte=neue_fruechte

# List Comprehension
fruechte=[frucht.strip() for frucht in fruechte]
```

Slicing

Aus einem Teilbereich einer Liste eine neue Liste erstellen.

```
s="abcdefg"

s[0]        # Erstes Zeichen: a
s[-1]       # Letztes Zeichen: g

# Zugriff auf Teile der Liste
s[:4]       # abcd
s[1:4]      # bcd
s[-2:]      # fg
```

Wichtige Methoden

```
a = [2,1,3]

len(a)            # Länge von a (hier 3)
a.append(3)       # a == [2,1,3,4]
del a[2]          # a == [2,1,4]
a.sort()          # a == [1,2,4]
```

Schleifen

if-Verzweigung

```
#!/usr/bin/python
import sys

print("Bitte Zahl eingeben:")
input=sys.stdin.readline()
input=input.strip() # Newline (\n) am Ende der Eingabe entfernen
i=int(input)

if i<0:
    print "%s ist kleiner als Null." %i
elif i==0:
    print "%s ist gleich Null." % i
else:
    print "%s ist größer als Null." % i
```

for-Schleife

```
for x in [1, 2, 3, 4]:
    print x

for x in range(6):
    if x==1:
        # Überspringe diese Zahl
        continue
    if x==4:
        # Beende die Schleife
        break
    print x
```

while-Schleife

```
running = 1
friends = []

while running:
    said = raw_input("Who's coming to the party? ")
    if said == "parents":
        print "Party plans abandoned"
        break
    if said == "":
        print "Data entry complete"
        running = 0
    else:
        friends.append(said)
else:
    print "Having a party for ",
    print friends

print "program exiting"
```

Module

Bibliotheken werden mit „import“ geladen.

Varianten

```
import sys  # lädt alle Funktionen aus sys in Namespace "sys"
            # Zugriff über Namespace-Qualifier "sys" nötig, z.B. sys.argv

from sys import exit # lädt nur die Funktion exit aus sys
                     # in aktuellen Namespace
from sys import *    # lädt alle Funktionen aus sys in aktuellen Namespace
```

Mehrere Module können auch gleichzeitig geladen werden:

```
import sys, os, re
```

Wichtige Module

- *os* und *os.path*: Systemnahe Funktionen wie Dateieigenschaften, anonyme Pipes etc.
- *re*: Suchen und Ersetzen regulärer Ausdrücke (vgl. THI)
- *math*: mathematische Funktionen (z.B. *cos*) und Konstanten (z.B. *pi*)
- *time*: Funktion *time.time()* gibt Zeitstempel zurück (für Laufzeitmessungen)

Funktionen

Funktionen werden durch das Schlüsselwort „def“ definiert.

```
def plus(a, b):
    return a+b

print plus(5, 7)  # 12
```

Parameter werden „Call By Reference“ übergeben. Das heißt die Parameter können innerhalb der Funktion geändert werden.

```
def fuelleListe(liste):
    liste.append("eins")
    liste.append("zwei")
    liste.append("drei")

liste=[]
fuelleListe(liste)
print liste            # ["eins", "zwei", "drei"]
```

Keyword Arguments, Argumente werden per Schlüsselwort übergeben.

```
def berechneWiderstand(volt, ampere):
    return float(volt)/ampere

print berechneWiderstand(volt=12, ampere=4)
print berechneWiderstand(ampere=4, volt=12)
```