003_MotivationPython

November 30, 2018

1 Warum Python?

1.0.1 klar verständlich

Wenn Sie ein Programm sehen und es gleich verstehen, dann ist es sehr wahrscheinlich Python

1.0.2 universell

- Versuche durchführen
- Meßgeräte steuern und auslesen
- Daten erheben, speichern, bearbeiten
- Auswerten
- Darstellen
  - interaktiv

1.0.3 mächtig

Bibliotheken...

- Linere Algebra
- Graphiken
- Bild-, Audio-, Daten-bearbeitung
- Statistik
- ...

1.0.4 weit verbreitet

- Linux, Unix, ...
  - Raspbian
- Mac
- Windows

1.0.5 open source

- https://docs.python.org/3/license.html
- Python Software Foundation (PSF, see https://www.python.org/psf/ , non-profit)
- GPL compatible

1.0.6 das Notebook

ipython und Jupyter als Arbeitsumgebung

Quelle Fernando Pérez, Brian E. Granger, IPython: A System for Interactive Scientific Computing, Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: http://ipython.org

Module:

- Mathematik
- Psychophysik
- Parallelrechnen (cuda)
- high sophisticated: Machine-Learning, MarkovChainsMonteCarlo, ICA ...
- Schnittstellen: serielle, Netzwerk, Webserver, Webbrowser

Notebook Leicht verständlich wie ich hier demonstrieren möchte:

2 Notebook

https://jupyter.org

Zellen

- code
- markdown
  - Sourcecode
  - Bilder
  - LaTeX: $a^2 = b^2 + c^2$
  - Listen (auch verschachtelt, self)
  - ...

Quelltext

- Ausführen
  - Python-Kernel
- Ausgabe, wird mit abgespeichert
  - Text
  - Graphik

3 Notebook

...

Exportieren

- ipynb
  - online Notebook-Viewer http://nbviewer.jupyter.org/
- Slideshow
- pdf

Hinweise

- umbenennen
- `m` markdown, `y` Quelltext
- merge/split
- Tastatursteuerung: Help%Shortcuts
- Aktive Kernel
- Save, Close&Halt

3.1 Einfache Datentypen

- integer
  - beliebige Präzision, automatisch
- float
  - *"Python's built-in str() function produces 12 significant digits"*
  - Normalerweise wird mit *17* Stellen gearbeitet
- boolean: True, False
- string
  - einfache (') oder doppelte (") Anführungszeichen
  - keine character

Alles sind Objekte

Vorsicht später

3.2 Kommentare

benutzen!!

```
In [1]: '''this is a comment'''

        '''triple-quote-comments can spread
           over several lines
           doc-strings are formatted like this
        '''

        r = 3     # create an integer object <3> and assign it to the name <r>
```

3.3 Zeichenketten

```
In [2]: # concatenate with '+', single or double quotes, possible to encapsulate
        print("String in 'single' " + 'or in "double" quotes')

        # include numbers in strings with '{}'.format() method:
        r = 1.23456
        print('r = {}'.format(r))

        # and format the numbers
        print('the square of {:.3f} is {:9.5f}'.format(r, r**2))
```

```
String in 'single'or in "double" quotes
r = 1.23456
the square of 1.235 is   1.52414
```

by the way, it's in English, because:

- syntax is somehow English anyhow
- doc is English
- technical terms most of the time are
- examples in web are
- if you publish, you will internationalize, too
- will be simple enough ;-)
- ask!

3.4 Zahlen

```
In [3]: x = 2                                              # integer
        print('x={}  is a {}  with object id {}'.format(x, type(x), id(x)))
```

```
x=2  is a <class 'int'>  with object id 140316490279488
```

```
In [4]: x = 2.6                                           # float
        print('x={}  is a {}  with object id {}'.format(x, type(x), id(x)))
```

```
x=2.6  is a <class 'float'>  with object id 140316282940512
```

```
In [5]: y = x
        print('x={:.16f}  is a {}  with object id {}'.format(x, type(x), id(x)))
        print('y={:.16f}  is a {}  with object id {}'.format(y, type(y), id(y)))
        y += 0.2
        print('y={:.16f}  is a {}  with object id {}'.format(y, type(y), id(y)))
        y -= 0.2
        print('y={:.16f}  is a {}  with object id {}'.format(y, type(y), id(y)))
        print('x and y have same value: {}'.format(x==y))
        print('x and y have same id:    {}'.format(x is y))
```

```
x=2.6000000000000001  is a <class 'float'>  with object id 140316282940512
y=2.6000000000000001  is a <class 'float'>  with object id 140316282940512
y=2.8000000000000003  is a <class 'float'>  with object id 140316282940488
y=2.6000000000000001  is a <class 'float'>  with object id 140316282940632
x and y have same value: True
x and y have same id:    False
```

```
In [6]: whos

Variable   Type     Data/Info
----------------------------
r          float    1.23456
x          float    2.6
y          float    2.6
```

- gleicher Variablenname
- nicht gleiches Objekt mit anderem Inhalt
- sondern neues Objekt und Referenz auf den Namen "x"
- interne Liste von Variablen:
  - who: Liste # nur in ipython/Jupyter
  - whos: mit Typ und Inhalt # nur -"-
  - dir(), globals(), locals() geben viel Brimborium aus wegen ipython-Umgebung

3.5 Tupel

- fassen Elemente zusammen, die zusammengehören
  - Bsp: x-y-Koordinatenpaar
  - Bsp: Ton = (Frequenz, Dauer, Amplitude, Name)
- Definition
  - getrennt durch Komma ',' (zwingend)
  - durch runde Klammern '(' und ')' (optional, Konvention)
- Elementzugriff
  - durch eckige Klammern '[' und ']'
  - Index muß Integer sein
  - Index beginnt bei 0 (wie in C)

```
In [2]: tone = (440.0, 250, 0.8, 'a')
```

```
In [3]: x = (1, 2, 3.)       # a tuple
        print('x={} and x is a {}'.format(x, type(x)))

x=(1, 2, 3.0) and x is a <class 'tuple'>
```

```
In [4]: n = len(x)                                          # length = number of elements in tuple
        print('first element={}, last={}'.format(x[0], x[n-1]))  # select two elements

first element=1, last=3.0
```

```
In [5]: print('x={} and x is a {} with elements {}, {} and {}'
              .format(x, type(x), type(x[0]), type(x[1]), type(x[2])))

x=(1, 2, 3.0) and x is a <class 'tuple'> with elements <class 'int'>, <class 'int'> and <class 'float'>
```

```
In [6]: d = (44, 55, x)                 # encapsulate tuple x within tuple d
        print(d)
        print(d[0])
        print(d[2][1])
```

```
(44, 55, (1, 2, 3.0))
44
2
```

```
In [13]: #%whos

         print(d[2][0])
```

```
1
```

- Zusammenfügen mit plus '+'

```
In [13]: y = 2.0, 3.14159, 66             # another tuple, (no brackets)
         z = x + y                        # concatenate two tuples
         print(x, ' + ', y, ' = ', z)     # it's not like adding two vectors!
```

```
(1, 2, 3.0)  +  (2.0, 3.14159, 66)  =  (1, 2, 3.0, 2.0, 3.14159, 66)
```

```
In [14]: print(2*y)          # multiple concatenation; not element-wise multiplication
```

```
(2.0, 3.14159, 66, 2.0, 3.14159, 66)
```

3.5.1 Tafel: z hinmalen, brauchen wir noch.

- Scheibenweise (*slicing*) mit Doppelpunkt ':'
  - Obacht: bis zum **vor**letzten Element
  - weglassen = erstes / letztes Element
  - negative Inicees zählen von hinten

```
In [15]: print(z[1:4])           # starting from #1 = 2nd, ending before 4th element
```

```
(2, 3.0, 2.0)
```

```
In [16]: print(z[2:-1])          # starting from #2 = 3rd, ending at last element
```

```
(3.0, 2.0, 3.14159)
```

```
In [17]: print(z[2:])            # starting from 3rd element; ending default is last
```

```
(3.0, 2.0, 3.14159, 66)
```

- Wie Messer: Schneiden **vor** dem Index
- Sprungweite hinter zweitem Doppelpunkt
  - Reihenfolge invertieren mit ":-1"

```
In [18]: print(z[::2])            # every 2nd element
```

```
(1, 3.0, 3.14159)
```

```
In [19]: print(z[2::-1])          # reverse order
```

```
(3.0, 2, 1)
```

Tupel unterscheiden sich von Vektoren/Matrizen. (später mehr)
Zur Erzeugung von Tupeln:

3.5.2 Einschub `for`-Schleife

```
In [20]: for i in d:          # key word "for" and colon
             print(i)         # .... indentation, default 4 chars
                              # that's it!
```

```
44
55
(1, 2, 3.0)
```

- Schlüsselwort "for"
- Doppelpunkt am Ende
- Einrückung allen Codes der Schleife (Standard 4 Spaces)
- anstatt von...bis reicht "in" und ein Objekt mit Inhalt

```
In [21]: for x in range(4):    # range creates a special object: from 0 on N elements (until last = N-1)
             print(x)
         print('after loop, x={}'.format(x))     # scope outside
```

```
0
1
2
3
after loop, x=3
```

```
In [22]: # if you need the index, then additionally:
         for i, a in enumerate(d):               # get index *and* elements
             print('d[{}] is {}'.format(i, a))   # still no need to index d[i]
```

```
d[0] is 44
d[1] is 55
d[2] is (1, 2, 3.0)
```

- range ist spezielles Aufzählungsobjekt
- ähnlich Tupel, ist aber keins
- wie Tupel-Indicees: von 0 ab N Stück, damit nur bis Index N-1
- Index von d nicht explizit nötig, a wird gleichzeitig extrahiert

```
In [23]: '''need more than one element simultaneously?'''
         a = (2, 4, 6, 10)
         b = (2, 5, 7, 13)
         for j, k in zip(a, b):      # like the zipper element-to-element (not zip like compact)
             print('b={:2d} belongs to a={:3d}'. format(k, j))
```

```
b= 2 belongs to a=  2
b= 5 belongs to a=  4
b= 7 belongs to a=  6
b=13 belongs to a= 10
```

3.5.3 Tupel erzeugen

```
In [23]: a = ()                         # empty tuple
         print('a={} is a {} and has {} elements'.format(a, type(a), len(a)))
         for i in range(1, 10, 2):      # odd numbers, from 1 to <10 step 2
             a += (i,)                  # tuple needs semicolon
         print('a={} is a {} and has {} elements'.format(a, type(a), len(a)))
```

```
a=() is a <class 'tuple'> and has 0 elements
a=(1, 3, 5, 7, 9) is a <class 'tuple'> and has 5 elements
```

```
In [24]: a[1] = 2        # you cannot change a tuple element


         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-24-17b5fa6fd3ad> in <module>()
     ----> 1 a[1] = 2        # you cannot change a tuple element


         TypeError: 'tuple' object does not support item assignment
```

3.6 Listen

- Listen sind Aufzählungstypen
  - oft, wenn auch nicht zwingend desselben Typs
- Listen werden von eckigen Klammern umgeben
- Listenelemente lassen sich ändern

fast alle Operationen wie bei Tupeln möglich

```
In [25]: l = [1, 2, 3, 4, 5.1]
         l[2] += 1
         print('l={} is a {} with {} elements; first of {}'.format(l, type(l), len(l), type(l[0])))

l=[1, 2, 4, 4, 5.1] is a <class 'list'> with 5 elements; first of <class 'int'>


In [26]: print('2*l+l:', 2*l+l)
         print('l[1:-1:2]', l[1:-1:2])

2*l+l: [1, 2, 4, 4, 5.1, 1, 2, 4, 4, 5.1, 1, 2, 4, 4, 5.1]
l[1:-1:2] [2, 4]


In [27]: '''careful with objects'''
         x = [1.2, 2.3, 3.4]
         y = x
         print('object id x    {}, y    {} '.format(id(x), id(y)))
         print('object id x[1] {}, y[1] {} '.format(id(x[1]), id(y[1])))
         y[1] = 0.2
         print('after y[1]={}  y[1]={}'.format(y[1], x[1]))
         print('object id x    {}, y    {} '.format(id(x), id(y)))
         print('object id x[1] {}, y[1] {} '.format(id(x[1]), id(y[1])))

object id x    140316184554056, y    140316184554056
object id x[1] 140316282941160, y[1] 140316282941160
after y[1]=0.2  y[1]=0.2
object id x    140316184554056, y    140316184554056
object id x[1] 140316184047736, y[1] 140316184047736
```

- Also Obacht ID ändert sich auch rückwärts, weil dasselbe Objekt innerhalb der List in selber Liste

3.6.1 Erzeugung: *list comprehension*

Mit eingebauter for-Schleife sehr elegant

```
In [28]: '''create list with internal for-loop'''
         squares = [x**2 for x in range(10)]    # list with built in for-loop
         print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
In [29]: '''multidimensional'''
         # inner loop: 10 elements x (0..9)
         # outer loop: 4 rows (power 0..3)
         powers = [[x**n for x in range(10)] for n in range(4)]   # list of lists
         for row in powers:
             print(row)
         i, j = (5, 3)                    # search for 5^3
         print('{}**{} = {}'.format(powers[1][i], j, powers[j][i]))
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
5**3 = 125
```

Ausblick Vektoren

```
In [30]: '''mathematical vector calculation'''
         squares = x**2           # squaring a vector?
```

```
         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-30-9e2eb4e24082> in <module>()
           1 '''mathematical vector calculation'''
     ----> 2 squares = x**2           # squaring a vector?


         TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

```
In [31]: '''matrix access'''
         x = powers[1, 2]
```

```
         ---------------------------------------------------------------------------

         TypeError                                 Traceback (most recent call last)

         <ipython-input-31-116512b94dfa> in <module>()
           1 '''matrix access'''
     ----> 2 x = powers[1, 2]


         TypeError: list indices must be integers or slices, not tuple
```

3.6.2 dictionary

- benannte Einträge
  - unterschiedliche Einträge möglich

```
In [32]: tone2 = {'freq': 440.0, 'duration': 250, 'ampl': 0.8, 'name': 'a'}
         print(tone[1], tone2['duration'])
```

```
250 250
```

```
In [33]: print(tone2[1])


        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-33-e030098c0e89> in <module>()
    ----> 1 print(tone2[1])


        KeyError: 1
```

```
In [34]: for key, value in tone2.items():    # access paired elements of dictionary
             print (key, value)
```

```
ampl 0.8
name a
freq 440.0
duration 250
```

```
In [35]: for key in tone2.keys():            # access keys only  of dictionary
             print('key name is {} and its value is {}'.format(key, tone2[key]))
```

```
key name is ampl and its value is 0.8
key name is name and its value is a
key name is freq and its value is 440.0
key name is duration and its value is 250
```

- ein dictionary ist weder Liste noch Tupel
- kein Zugriff auf einzelne Paare

4 numpy : mathematische Bibliothek

- Link: http://www.numpy.org/
- Open source Lizenz
  - BSD-new license (3-clause) http://www.numpy.org/license.html
- Installieren
  - apt-get install python-numpy
  - conda install numpy
  - https://www.scipy.org/install.html
- Importieren
  - Namenskonvention für Schreibfaulheit np

```
In [36]: import numpy as np      # allow access to all numpy functions, datatypes, ...

         '''numpy vector'''
         v = np.array(x)         # make numpy vector from previous list-object
         print('v={} is a {} with {} elements of {}'.format(v, type(v), len(v), type(v[0])))
```

```
v=[ 1.2  0.2  3.4] is a <class 'numpy.ndarray'> with 3 elements of <class 'numpy.float64'>
```

```
In [37]: '''vector maths'''
         sq = v**2               # now we can calculate with vectors (numpy arrays)
         print('sq={} is a {} with {} elements of {}'.format(sq, type(sq), len(sq), type(sq[0])))
```

```
sq=[  1.44   0.04  11.56] is a <class 'numpy.ndarray'> with 3 elements of <class 'numpy.float64'>
```

```
In [38]: print('norm |v|={}'.format(np.dot(v, v) ))   # scalar product
```

```
norm |v|=13.04
```

```
In [39]: print('v has {} values of {} with a mean={:.3f} and sd={:.3f}'.
              format(v.shape[0], v.dtype, v.mean(), v.std()))
```

```
v has 3 values of float64 with a mean=1.600 and sd=1.337
```

```
In [40]: print('the maximum of v is {:.3f} located at index {}'.format(v.max(), v.argmax()))
```

```
the maximum of v is 3.400 located at index 2
```

```
In [41]: '''matrix'''
         po = np.asarray(powers)
         print('po=\n{} \nis a {} with shape {} of {}'.format(po, type(po), po.shape, type(po[0, 0])))
```

```
po=
[[  1   1   1   1   1   1   1   1   1   1]
 [  0   1   2   3   4   5   6   7   8   9]
 [  0   1   4   9  16  25  36  49  64  81]
 [  0   1   8  27  64 125 216 343 512 729]]
is a <class 'numpy.ndarray'> with shape (4, 10) of <class 'numpy.int64'>
```

```
In [42]: '''matrix calculation'''
         A = np.array([[1, 2, 3], [3, 2, 1], [3, 3, 0]])    # matrix
         x = np.array([5, 6, 10])                           # vector
         y = np.dot(A, x)                                   # multiplication
         print(y)                                           # result, always a row-vector
```

```
[47 37 33]
```

```
In [43]: '''matrix functions'''
         Ai = np.linalg.inv(A)    # sub module linalg of numpy provides matrix inversion
         z = np.dot(Ai, y)        # test inversion
         print(z-x)               # accuracy is about 15 decimal digits
```

```
[  0.00000000e+00   3.55271368e-15   0.00000000e+00]
```

```
In [44]: '''inverse?'''
         E = np.dot(A, Ai)
         print(E)                                    # we expect 1
         print(np.all(np.isclose(E, np.eye(3))))    # every element within given precision?
```

```
[[  1.00000000e+00   4.44089210e-16   0.00000000e+00]
 [  0.00000000e+00   1.00000000e+00  -1.11022302e-16]
 [  0.00000000e+00   4.44089210e-16   1.00000000e+00]]
True
```

```
In [45]: '''matrix chains'''
         print(A.dot(A.dot(x)))   # easy to interpret, transposes
```

```
[220 248 252]
```

```
In [46]: '''zero matrix'''
         np.zeros((3, 5))   # outer 3 rows x inner 5 columns
```

```
Out[46]: array([[ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.],
                [ 0.,  0.,  0.,  0.,  0.]])
```

```
In [47]: print(np.ones_like(A))   # all elements 1; in shape of A, that means 3x3
```

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

```
In [48]: print('{} is the transposed of A'.format(A.T))
```

```
[[1 3 3]
 [2 2 3]
 [3 1 0]] is the transposed of A
```

- Matrix aus Liste von Listen (gleicher Länge natürlich)
- Zugriff auf einzelnes Element mit 2 Indices möglich po[0, 0]
- dot: Vektor/Matrix-Multiplikation
  - alle Vektoren sind Zeilenvektoren
  - dot interpretiert als Spaltenvektor.

```
In [49]: '''all 1d numpy ndarrays are row-vectors.
            numpy.dot transposes row-vector to column-vector, multiplicates
            and returns row-vector'''
         Ma = np.ones((3,5))
         Va = np.ones(5)
         print(Ma)
         print(Va)
         print(np.dot(Ma, Va))
```

```
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
[ 1.  1.  1.  1.  1.]
[ 5.  5.  5.]
```

```
In [50]: print('flatten A: {}'.format(A.ravel()))
```

```
flatten A: [1 2 3 3 2 1 3 3 0]
```

```
In [51]: print('{} reshaped from {} values'.format(np.arange(3*4).reshape((3, 4)), 3*4))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]] reshaped from 12 values
```

```
In [52]: '''slicing'''
         print(A[:-1, 1:])   # skip last row; skip first column
```

```
[[2 3]
 [2 1]]
```

```
In [53]: '''concatenate column or row'''
         print('{} A with additional row'.format(np.vstack((A, [[5, 6, 7]]))))
         print('{} A with additional column'.format(np.hstack((A, [[4], [0], [0]]))))
```

```
[[1 2 3]
 [3 2 1]
 [3 3 0]
 [5 6 7]] A with additional row
[[1 2 3 4]
 [3 2 1 0]
 [3 3 0 0]] A with additional column
```

Literatur:

- https://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html
- https://docs.scipy.org/doc/numpy/reference/routines.array-manipulation.html
- https://docs.scipy.org/doc/numpy/reference/routines.linalg.html
- https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html

4.1 Funktionen

- Definition
  - Schlüsselwort "def"
  - Doppelpunkt nach Definitionszeile
  - Argumente in Klammern
  - eingerückter Quelltext (Standard 4 Leerzeichen)

```
In [54]: def square(a):
             return a**2

         print(square(4))
```

```
16
```

4.1.1 Bitte Kommentare!

```
In [55]: def square(a):
             '''
             function my_func

             Parameters
             ----------
             a : int, float or numpy-array

             Returns
             -------
             y : a^2; element wise square of a

             Examples
             --------
             >>>my_func(90)
             8101

             Note
             ----
             useful for showing how to comment a function with docstring
             '''
             return a**2
```

```
In [56]: square?
```

```
In [58]: '''recursion'''
         def d( a ):
             if len(a)>1:                          # finished?
```

```
                print('a {} is '.format(type(a)))
                d(a[1])                             # go further ...
            else:
                print('a {}'.format(type(a)))       # result

        mylist = [0, [1, [2, [3]]]]
        d(mylist)
```

```
a <class 'list'> is
a <class 'list'> is
a <class 'list'> is
a <class 'list'>
```

- verschachtelte Einrückung
- Rekursion

Argument beim Namen nennen

```
In [1]: def myfunc(a1, a2, a3):
            print('erstes {}, zweites {} und drittes {} Argument'.format(a1, a2, a3))

        myfunc(1, 2, 3 )
```

```
erstes 1, zweites 2 und drittes 3 Argument
```

```
In [2]: myfunc(a2=20, a3=30, a1=10)
```

```
erstes 10, zweites 20 und drittes 30 Argument
```

default Argumente

```
In [3]: def myfunc(a1=1, a2=2, a3=3):
            print('erstes {}, zweites {} und drittes {} Argument'.format(a1, a2, a3))

        myfunc(a2=22)
```

```
erstes 1, zweites 22 und drittes 3 Argument
```

4.2 Verzweigung

```
In [59]: '''if-then-elif-else'''
         a = 3.1
         b = 3.141
         if a==b:
             print('  equal')
         elif a>b:
             print('  greater')
         else:
             print('- the else case: -')
             print('  different')
```

```
- the else case: -
  different
```

Vergleichsoperatoren

```
<, >, ==, >=, <=, !=      # gleiche Priorität, unterhalb Arithmetik
```

Siehe: https://docs.python.org/3/reference/expressions.html#comparisons

```
In [60]: '''while loop'''
         l = [1, 2, 3, 4]
         while len(l):          # 0 is equivalent to False, any other to True
             print('this 0-element={}'.format(l[0]))
             l = l[1:]

         print('rest of l: {}'.format(l))
```

```
this 0-element=1
this 0-element=2
this 0-element=3
this 0-element=4
rest of l: []
```

Kann meistens eleganter durch for ersetzt werden durch Verwendung von in: for x in list:

```
In [61]: '''do-while replacement'''
         a = 'y'

         while a != 'x':
             a = input('Type something <ENTER>, "x" stops: ')
             print('you pressed {}'.format(a))

         print('done')
```

```
Type something <ENTER>, "x" stops: A
you pressed A
Type something <ENTER>, "x" stops: B
you pressed B
Type something <ENTER>, "x" stops: C
you pressed C
Type something <ENTER>, "x" stops: x
you pressed x
done
```

switch-case ist nicht in Python implementiert.
Ersatz durch if-elif-else oder siehe etwa hier: http://code.activestate.com/recipes/410692/

4.3 matplotlib Graphiken

- Projekthomepage http://matplotlib.org/index.html
- Beispiele http://matplotlib.org/gallery.html
- Einführung http://matplotlib.org/users/pyplot_tutorial.html

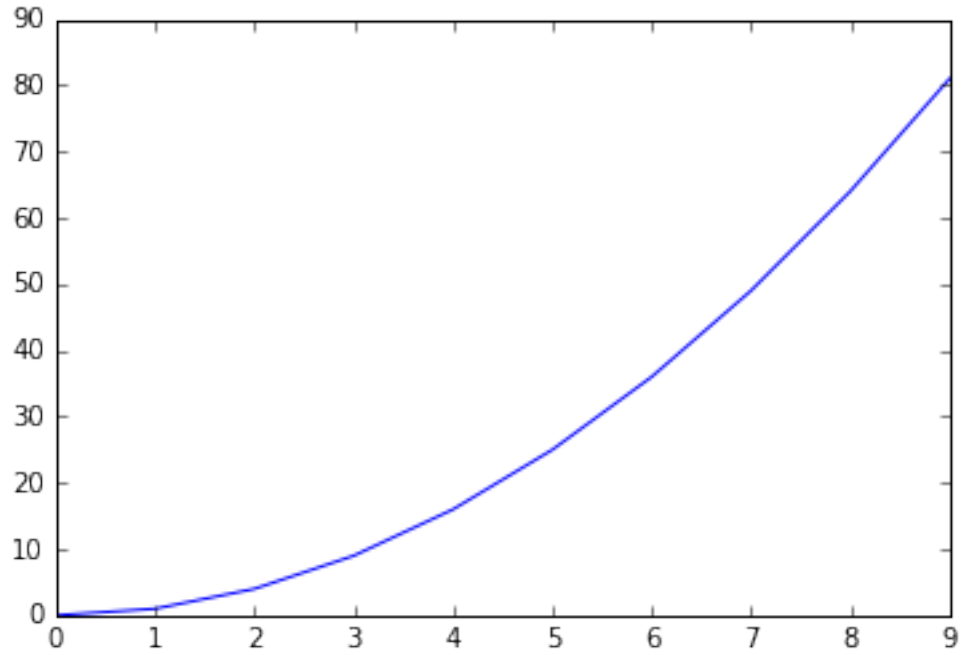
Lizenz

- based on the Python Software Foundation (PSF) license

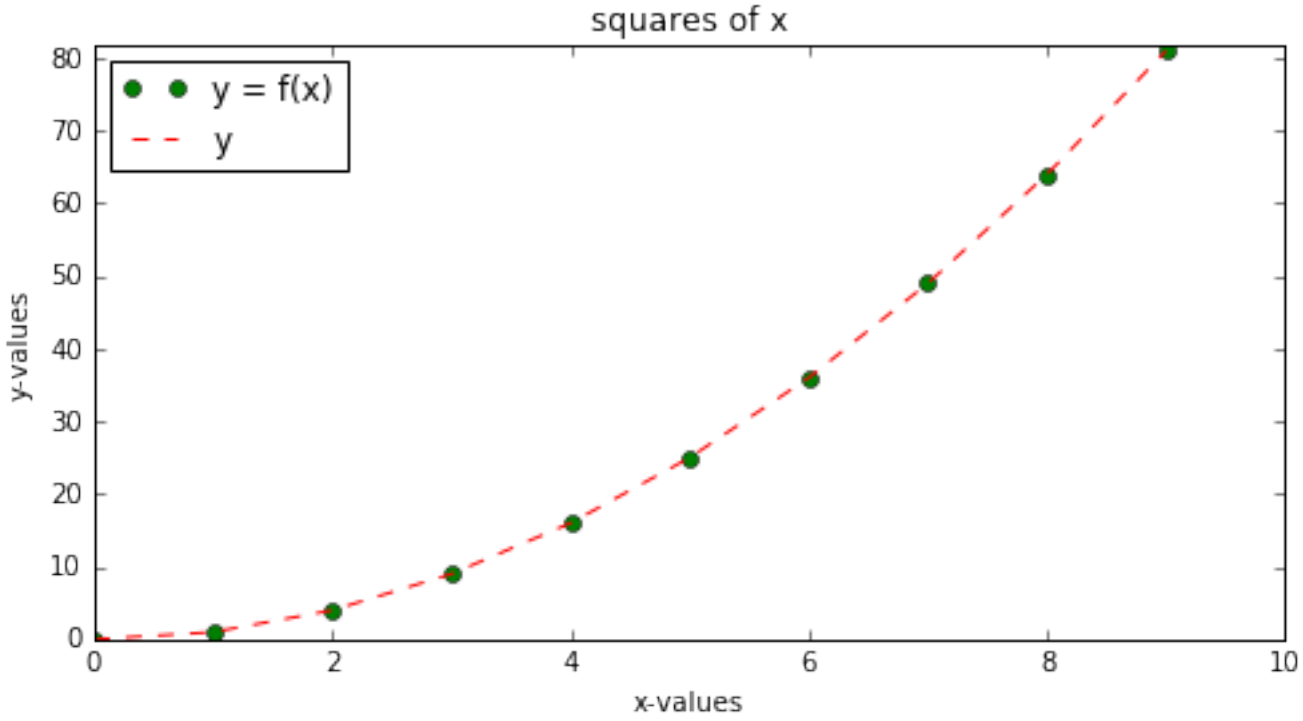
```
In [62]: '''add graphics-(sub)-library, and set as output within notebook:'''
         from matplotlib import pyplot as plt
         %matplotlib inline
```

```
In [63]: '''define x and y and plot y vs x'''
         x = powers[1]
         y = powers[2]
         plt.plot(x,y)
```

Out[63]: [<matplotlib.lines.Line2D at 0x7f9dc4d74d30>]

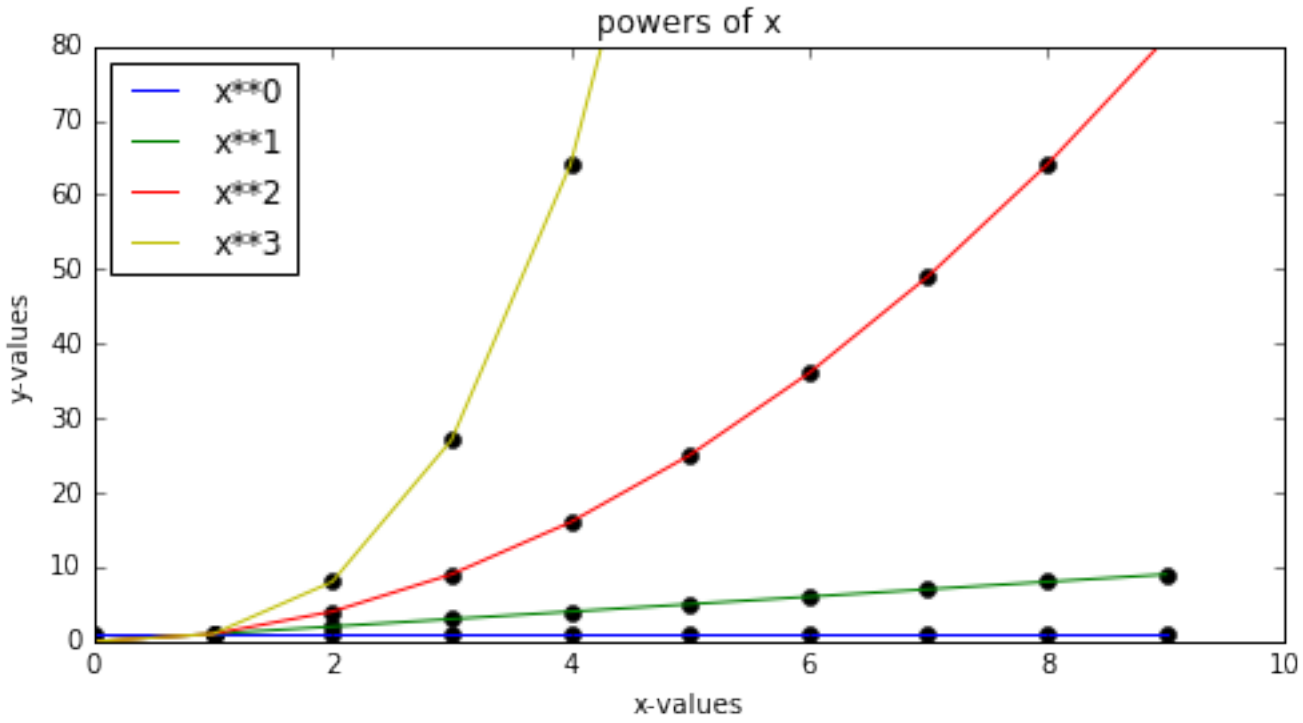


In [64]:
```
'''plot it nicely'''
fig = plt.figure(figsize=(8, 4))          # create a figure with size 8x4 inches
plt.title('squares of x')                 # write a headline on top of the figure
plt.xlabel('x-values')                    # write x-axis' label
plt.ylabel('y-values')                    # write y-axis' label
plt.axis([0, 10, 0, 82])                  # restrict x- and y-axis
plt.plot(x, y, 'go', label='y = f(x)')    # plot y vs x; use green circles; set plotlabel
plt.plot(x, y, 'r--', label='y')          # ...  and also interpolating red dashed lines
plt.legend(loc='upper left');             # ';' prevents object output
```



```
In [65]: '''plot nicely multiple graphs in different colors'''
         fig = plt.figure(figsize=(8, 4))          # create a figure with size 8x4 inches
         x = powers[1]
         plt.title('powers of x')                  # write a headline on top of the figure
         plt.xlabel('x-values')                    # write x-axis' label
         plt.ylabel('y-values')                    # write y-axis' label
         plt.axis([0, 10, 0, 80])                  # restrict x- and y-axis
         plt.plot(x, np.asarray(powers).T, 'ko')   # plot all 4 ys vs x at once; use black circles
         for i, (row, col) in enumerate(zip(powers, ['b', 'g', 'r', 'y'])):
             print("plotting x to the power of {} in color '{}'".format(i, col))
             plt.plot(x, row, color=col, linestyle='-', label='x**{}'.format(i))
         plt.legend(loc='upper left');
```

```
plotting x to the power of 0 in color 'b'
plotting x to the power of 1 in color 'g'
plotting x to the power of 2 in color 'r'
plotting x to the power of 3 in color 'y'
```



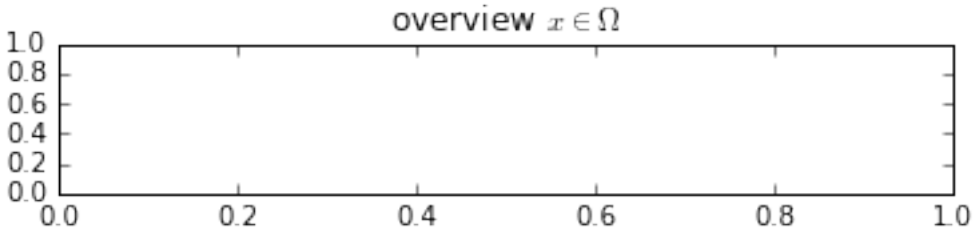
- Verschönern:
  - Titel
  - Achsen
  - Legende mit Beschriftungen
  - Punkte
  - Verbinden
- der ';' verhindert die Ausgabe des Objekts

4.3.1 LaTeX

http://matplotlib.org/users/mathtext.html

```
In [66]: '''LaTeX-math is working under graphics matplotlib
            see http://matplotlib.org/users/mathtext.html'''
         import matplotlib.pyplot as plt          # enable graphics; within notebook:
         %matplotlib inline
```

```
plt.figure(figsize=(6,1))
plt.title(r'overview $x \in \Omega$');  # r=raw string; $...$ frames math
```

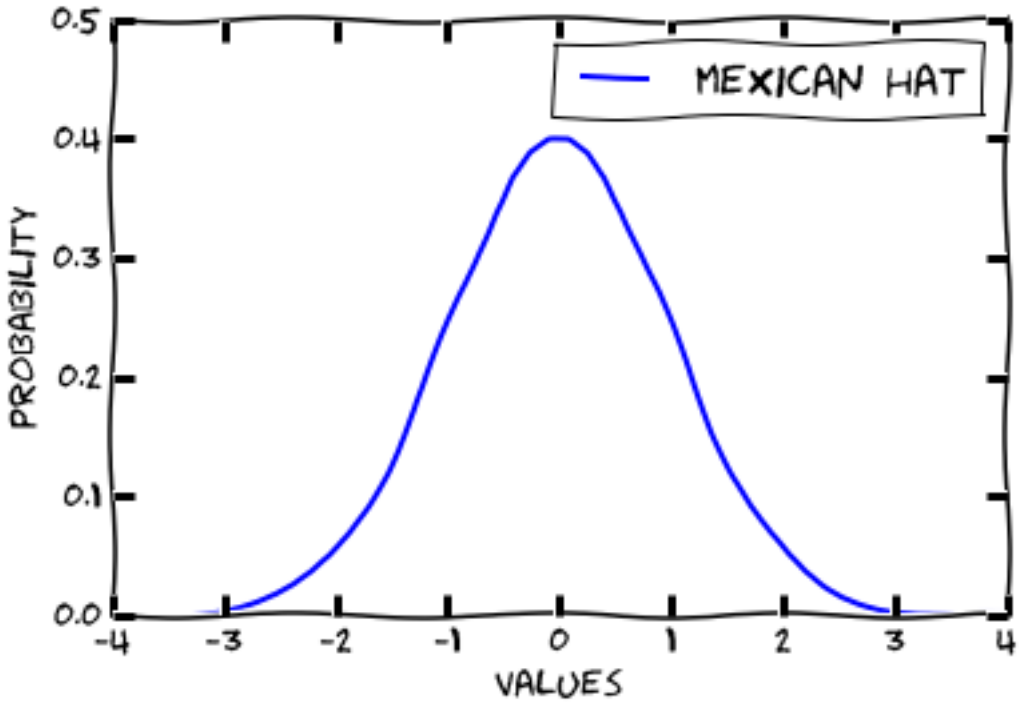


4.4 Statistik

Siehe http://docs.scipy.org/doc/scipy/reference/stats.html

Wir beschäftigen uns *ausgiebig* mit Statistik. Hier nur ein keines Beispiel, alles weitere wenn erforderlich.

```
In [67]: '''plot a Gaussian distribution'''
         from scipy import stats              # statistics library in scientific library
         x = np.linspace(-4, 4)               # linear spaced x-values in a numpy array, default 50
         plt.xkcd()                           # just call XKCD-style
         # plot the famous normal probability density function
         plt.plot(x, stats.norm.pdf(x), label='mexican hat')
         plt.axis((-4, 4, 0, 0.5))            # restrict x and y
         plt.xlabel('values')
         plt.ylabel('probability')
         plt.legend(loc='upper right');
```



4.5 Betriebssystem

```
In [68]: print('-- my System:')
         ! uname -mr
         ! python --version
         print('-- where am i?')
         ! pwd
         print('-- some python files:')
         ! ls *.py
         print('-- hello.py contains:')
         ! cat hello.py
```

```
-- my System:
3.10.0-327.36.2.el7.x86_64 x86_64
Python 3.5.2 :: Continuum Analytics, Inc.
-- where am i?
/home/wannek/VL/statistik
-- some python files:
getch_half.py  getch.py  hello.py  Kruschke.py
-- hello.py contains:
print('hello world')
```

```
In [4]: print('-- my System:')
        ! uname -mr
        ! python --version
        print('-- where am i?')
        ! pwd
        print('-- some python files:')
        ! ls *.py
        print('-- hello.py contains:')
        ! cat hello.py
```

```
-- my System:
4.4.140-1.el7.elrepo.x86_64 x86_64
Python 3.7.0
-- where am i?
/home/wannek/VL/statistik
-- some python files:
AngStII_Bayes.py  getch_half.py  getch.py  hello.py
-- hello.py contains:
print('hello world')
```

4.6 Line magics %

https://ipython.org/ipython-doc/dev/interactive/magics.html

```
%load hello.py
```

```
In [69]: # %load hello.py
         print('hello world')
```

```
hello world
```

- Erster Aufruf lädt (und setzt Kommentar)
- Zweiter Aufruf führt aus

```
In [70]: %run hello.py
```

```
hello world
```

```
In [71]: %%writefile hello2.py
         print('hello world #2')

Writing hello2.py


In [72]: ! cat hello2.py

print('hello world #2')

In [73]: %matplotlib inline
```

Export als *notebook* aus *ipython* heraus:

```
%notebook -e FILENAME
```

- Falls Sie *nicht* **jupyter notebook** verwenden, können Sie so auch ein Notebook erstellen.

```
In [74]: %pinfo A

Type:            ndarray
String form:
[[1 2 3]
 [3 2 1]
 [3 3 0]]
Length:          3
File:            /home/data/anacondaCent/envs/statistik/lib/python3.5/site-packages/numpy/__init__.py
Docstring:       <no docstring>
Class docstring:
ndarray(shape, dtype=float, buffer=None, offset=0,
        strides=None, order=None)

An array object represents a multidimensional, homogeneous array
of fixed-size items.  An associated data-type object describes the
format of each element in the array (its byte-order, how many bytes it
occupies in memory, whether it is an integer, a floating point number,
or something else, etc.)
```

5 Hilfe

- In *ipython / jupyter* funktioniert die **automatische TAB**-Vervollständigung.
- Aufruf eines Befehls mit anghängtem **Fragezeichen** zeigt die Hilfe in einem extra Fenster.
- `help()`-Funktion
- Mittels print die **doc-Strings** anzeigen .

- `print( .__doc__)` auch in ipython Textkonsole fortlaufend im Text
- `help()`
  - in jupyter-code-Zelle dann in der Ausgabe
  - in ipython als Editor-Frame überlagert
- `?` in jupyter notebook als dauerhaftes Info-Fenster (schließbar)

```
>>> from scipy import stats
>>> print( stats.norm.__doc__ )
A normal continuous random variable.
  The location (loc) keyword specifies the mean.
  The scale (scale) keyword specifies the standard deviation.
  Continuous random variables are defined from a standard form and may
  require some shape parameters to complete its specification.  Any
  optional keyword parameters can be passed to the methods of the RV
  object as given below:

  Methods
  -------
```

```
  ``rvs(loc=0, scale=1, size=1)``
      Random variates.
  ``pdf(x, loc=0, scale=1)``
      Probability density function.
  ``logpdf(x, loc=0, scale=1)``
      Log of the probability density function.
  ``cdf(x, loc=0, scale=1)``
      Cumulative density function.
[...]

>>> print( stats.norm.pdf.__doc__ )
Probability density function at x of the given RV.

  Parameters
  ----------
  x : array_like
      quantiles
  arg1, arg2, arg3,... : array_like
      The shape parameter(s) for the distribution (see docstring of the
      instance object for more information)
  loc : array_like, optional
      location parameter (default=0)
  scale : array_like, optional
      scale parameter (default=1)
  Returns
  -------
  pdf : ndarray
      Probability density function evaluated at x

help(plt.scatter)

Help on function scatter in module matplotlib.pyplot:

scatter(x, y, s=20, c=None, marker='o', cmap=None, norm=None, vmin=None, vmax=None,
        alpha=None, linewidths=None, verts=None, edgecolors=None, hold=None, data=None,
        **kwargs)
Make a scatter plot of x vs y, where x and y are sequence like objects
of the same lengths.

Parameters
----------
x, y : array_like, shape (n, )
    Input data

s : scalar or array_like, shape (n, ), optional, default: 20
    size in points^2.

c : color or sequence of color, optional, default : 'b'
    `c` can be a single color format string, or a sequence of color
    specifications of length `N`, or a sequence of `N` numbers to be
    mapped to colors using the `cmap` and `norm` specified via kwargs
    (see below). Note that `c` should not be a single numeric RGB or
    RGBA sequence because that is indistinguishable from an array of
    values to be colormapped.  `c` can be a 2-D array in which the
    rows are RGB or RGBA, however, including the case of a single
    row to specify the same color for all points.

marker : `~matplotlib.markers.MarkerStyle`, optional, default: 'o'
    See `~matplotlib.markers` for more information on the different
[...]
```

5.1 Links

THE ipython cookbook. Code available. http://ipython-books.github.io/cookbook/

Einführung! http://nbviewer.ipython.org/github/gestaltrevision/python_for_visres/blob/master/index.ipynb

Natural Image Statistics http://nbviewer.ipython.org/github/gestaltrevision/python_for_visres/blob/master/Part7/Part7_Ima

Sammlung Psych/Neuro: https://github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks#psychology-and-neuroscience

Nice graph-examples and options http://www.labri.fr/perso/nrougier/teaching/matplotlib/, http://www.ster.kuleuven.be/~pieterd/python/html/plotting/matplotlib.html

Nice plots with different libraries, eg. density kernel plot, makes use of skripting: http://nbviewer.ipython.org/gist/msund/7ac1203ded66fe8134cc

scipy.optimize mit Psychometrischer Funktion http://nbviewer.ipython.org/github/arokem/teach_optimization/blob/master/c

Für Matlab Umsteiger: https://docs.scipy.org/doc/numpy/user/numpy-for-matlab-users.html

http://mathesaurus.sourceforge.net/matlab-numpy.html

6 Fragen?