Programmieren mit Python

Comprehensions

Listen/Mengen

$$A = \{\, x + 1 \mid x \in [1, 10) \,\}$$
$$B = \{\, x + 1 \mid x \in A,\ x = 2 \mod 3 \,\}$$

```
>>> a = {x + 1 for x in range(1, 10)}
>>> b = {x + 1 for x in a if x % 3 == 2}
>>> a
{2, 3, 4, 5, 6, 7, 8, 9, 10}
>>> b
{3, 6, 9}
```

Verschachtelung ist möglich:

```
>>>[{x for x in range(1,y+1) if x <= y} for y in range(1,6)]
[{1}, {1, 2}, {1, 2, 3}, {1, 2, 3, 4}, {1, 2, 3, 4, 5}]
```

Sequenzen

- Auch für Sets und Dictionaries
- Faulheit: Generator-Ausdrücke

```
>>> {x % 3 for x in range(1, 10)}
{0, 1, 2}
>>> {w: len(w) for w in ['eggs', 'bacon', 'spam']}
{'spam': 4, 'eggs': 4, 'bacon': 5}
>>> (x % 3 for x in range(1, 1000))
<generator object <genexpr> at 0x7fe12484be10>
>>> len(list(_))
999
```

Programmieren mit Python

Module

Import

```
>>> import math
>>> math.sin(0)
0.0
>>> import math as m
>>> m.sin(0)
0.0
>>> from math import sin, cos
>>> sin(0)
0.0
>>> from math import sin as s
>>> s(0)
0.0
>>> from math import *
>>> tan(0)
0.0
```


Batteries included

Nützliche Module in der Standardbibliothek

- Textverarbeitung: `re`, `string`, `textwrap`
- Mathematik: `decimal`, `fractions`, `math`, `random`, `statistics`
- Funktionen: `functools`, `itertools`, `operator`
- Sonstiges: `argparse`, `logging`, `multiprocessing`, `os`
- Datascience: `NumPy`, `SciPy`, `Pandas`, `Matplotlib`, `Seaborn`, `SciKit-Learn`,...
- Ostereier: `antigravity`, `this`
- ...

Exkurs: Hausaufgabe

```python
from math import log, ceil
from operator import mul
from functools import reduce

def factorial(n):
    return reduce(mul, range(1, n + 1))

def digit_sum(n):
    def digit(n, idx):
        return n // 10 ** idx % 10

    def digits(n):
        return range(0, ceil(log(n, 10)) + 1)

    return sum([digit(n, i) for i in digits(n)])
```

Programmieren mit Python

Exceptions

Fehlerbehandlung

```
try:
    print(1 // int(input('> i = ')))
except ZeroDivisionError:
    print('Division by zero')
except Exception as e:
    print(repr(e))
else:
    print('no error')
```

```
> i = 0
Division by zero
> i = a
ValueError("invalid literal for int() with base 10: 'a'",)
> i = 1
1
no error
```

Programmieren mit Python

IO

Kontext

- open liefert `File`-Instanz
- Automatisches Aufräumen

```
>>> with open('/etc/issue.net') as issue:
...     print(issue.read())
...
Debian GNU/Linux 8
```


Schreiben

- Ausgabe analog

```
>>> with open('spam.txt', mode='w') as spam:
...     for i in range(1, 5):
...         spam.write('{}\n'.format(i))
...
2
2
2
2
```

Programmieren mit Python

Operatorüberladung

Am Beispiel I

```
1  class Point:
2      def __init__(self, x = 0, y = 0):
3          self.x = x
4          self.y = y
5
6      def __str__(self):
7          return "({0},{1})".format(self.x,self.y)
8
9      def __add__(self,other):
10         x = self.x + other.x
11         y = self.y + other.y
12         return Point(x,y)
```

```
1  >>> p1 = Point(3,4); p2 = Point(-2,3)
2  >>> print(p1 + p2)
3  (1,7)
```

Am Beispiel II

```
1  class Point:
2      def __init__(self, x = 0, y = 0):
3          self.x = x
4          self.y = y
5
6      def __lt__(self,other):
7          return self.x < other.x
```

```
1  >>> p1 = Point(3,4); p2 = Point(-2,3)
2  >>> print(p1 < p2)
3  (1,7)
```

How and why?

- Keine (wirkliche) Thread-Parallelität wegen GIL (Global Interpreter Lock)
- ⇒ Kein Multithreading!
- ABER: Außerhalb von GIL kann „jeder machen was gewollt“
- ⇒ Process-Paralellität
- Zum Beispiel `NumPy` tut das!
- ABER: Nur Sinnvoll ab Mindestgröße

Process	**Threads**
no shared memory	shared memory
spawning expensive	spawning less expensive
memory sync not needed	memory sync needed

einfaches Beispiel (aus Python-doc)

```
1 from multiprocessing import Pool, TimeoutError
2 import time
3 import os
4
5 def f(x):
6     return x*x
7 with Pool(processes=4) as pool:
8         # print "[0, 1, 4,..., 81]"
9         print(pool.map(f, range(10)))
```

Wissenschaftliches Arbeiten mit Python

Ziele des Abschnitts

Ziele

- Vorstellung von „Standard-Bibliotheken“
- Vorstellung des „Python Scientific Computing Environments“ (SciPy)

SciPy, was ist das? Ein großes Konglomerat von Python-Bibliotheken zum:

- **Visualisieren** von verschiedensten Zusammhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- Lösen von **Optimierungsproblemen**.
- **Datenanalyse**, zum Beispiel mittels importierten Paketen von `R`.
- Nutzen von **symbolischer Mathematik**
- Bereitstellen von besseren, an die zu lösenden Probleme angepasste, **interaktiven Interpretern**.
- Bereitstellen von Tools die dem jeweiligen **Wissenschaftsfeld angepasst sind**.
- Anbindung an **schnelle** Programmiersprachen wie Fortran 95 oder C++.

Wissenschaftliches Arbeiten mit Python

Numerisches Rechnen (NumPy)

Was ist NumPy?

- Spracherweiterung zu Python
- Anwendungsfeld ist (schnelles) numerisches Rechnen
- stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung
- lagert Rechnungen an schnelle Bibliotheken BLAS[1] und LAPACK[2] aus.

Woher bekomme ich es?

- In Debian-ähnlichen OS: `aptitude install python3-numpy`
- Die neueste Version ist erhältlich unter:
  `git clone git://github.com/numpy/numpy.git numpy`
- Am besten via `pip3 install --user numpy`

Wie aktiviert man es?

```
1 import numpy as np
```

[1] Basic Linear Algebra Subprograms
[2] Linear Algebra Package

Einschub Package-installer (pip)

- Python besitzt seinen eigenen Package-installer pip.
- Dieser ermöglicht unabhängig vom Betriebssystem Python-Pakete zu installieren aus dem PyPI (Python Package Index).

Howto PyPi

- `sudo apt-get install python3-pip` (einmalig)
- danach steht das Kommando `pip3` zur Verfügung
- `pip3 install --user 'SomePackage'`, zum Beispiel: `pip3 install 'numpy'`

Für viele python-packages wird das Betriebssystempaket `python3-dev` vorausgesetzt, also `sudo aptitude install python3-dev`.

Los geht's NumPy: Arrays („To the n-th dimension“)

```
1  import numpy as np; import math as ma
2  A = np.arange(12).reshape(3, 4)
3  print(A); type(A)
4  A.dtype
5
6  B = A * ma.pi
7  print(B); type(B)
8  B.dtype
9
10 SomeList = [1.2, 3.4, 7.8, 9.6, 10.4, 7.4, 9.9, 3.3, 2.2]
11 C = np.array(SomeList).reshape(3,3); v = np.array([1,2,3])
12 C * v #?
13 C.dot(v)
```

NumPy: Matrices („To the 2nd dimension“)

```
1  import numpy as np;
2  A = np.arange(9).reshape(3, 3)
3  M = np.matrix(A)
4  type(M)
5  print(A*A)
6  print(M*M)
7
8  print(M.I)  # matrices have inverse by default
9  v= np.matrix([[1],[2],[3]])
10 print(M*v)
```

Die **matrix**-Klasse bietet also eine einfach und gewohnte Notation für Standardoperationen.

NumPy: Lineare Algebra

```
1  import numpy as np; import numpy.linalg as npalg
2  M = np.matrix([1,2,0,2,0,0,0,3,3]).reshape(3, 3)
3  v = np.matrix([[1],[2],[3]])
4
5  print(npalg.det(M))
6
7  print(npalg.eigvals(M))
8
9  print(npalg.eig(M)) # Eigenvalues and (right) eigenvectors
10 print(npalg.solve(M,v))
```

Für alles was man brauchen könnte gibt es eine methode.

http://docs.scipy.org/doc/numpy/reference/routines.linalg.html

Common mistake!

```
>>> a = np.array(1,2,3,4)     # WRONG
>>> a = np.array([1,2,3,4])   # RIGHT
```

Reshape!

```
>>> b = np.arange(12).reshape(4,3); print(b) # 2d array
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
>>> c = np.arange(24).reshape(2,3,4); print(c) # 3d array
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]
 [[12 13 14 15]
  [16 17 18 19]
  [20 21 22 23]]]
```

Wissenschaftliches Arbeiten mit Python

It's all about that (drawing) space: *Matplotlib*

matplotlib, pyplot and pylab?!?

- `matplotlib` ist das „whole package“
- `pyplotb` ist ein Modul in `matplotlib`, vorzugsweise zu verwenden für nicht-interaktive Plots.
- `pylab` ist vorzugsweise in interaktiven Berechnungen/Plots zu verwenden.

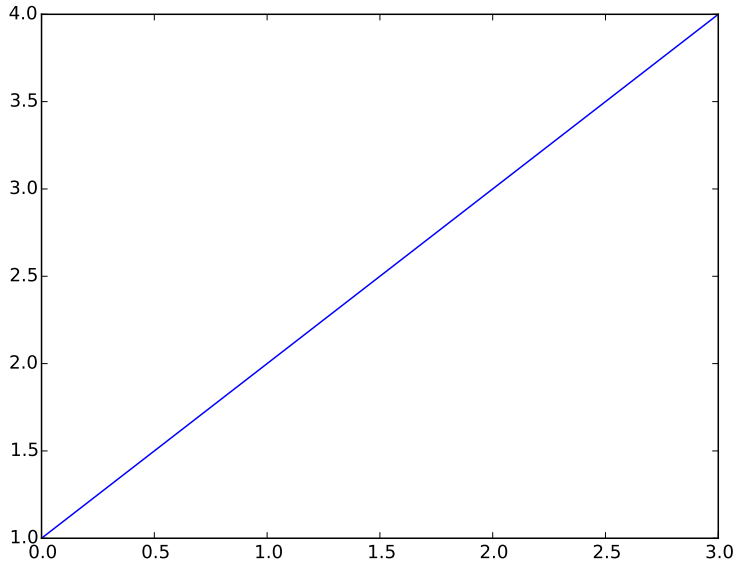
Viele Beispiele zu Plots auf dieser und auf den nächsten Folien wurden `http://matplotlib.org/users/screenshots.html` entnommen.

Nutzung von matplotlib und Python3 unter Debian

`sudo aptitude install tk-dev`,
`sudo aptitude install python-tk`, und danach
`pip3 install --user matplotlib`.

Simple plot (code)

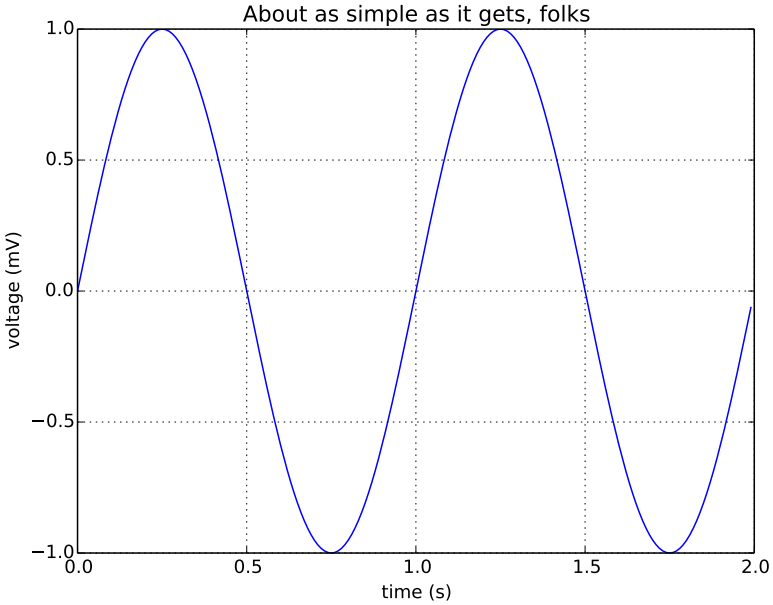
```
1 import matplotlib.pyplot as plt
2 plt.plot([1,2,3,4])
3 plt.show()
```



Little less simple plot (code)

```
1  from pylab import *
2
3  t = arange(0.0, 2.0, 0.01)
4  s = sin(2*pi*t)
5  plot(t, s)
6
7  xlabel('time␣(s)')
8  ylabel('voltage␣(mV)')
9  title('About␣as␣simple␣as␣it␣gets,␣folks')
10 grid(True)
11 savefig("test.png")
12 show()
```


Little less simple plot (Bild)



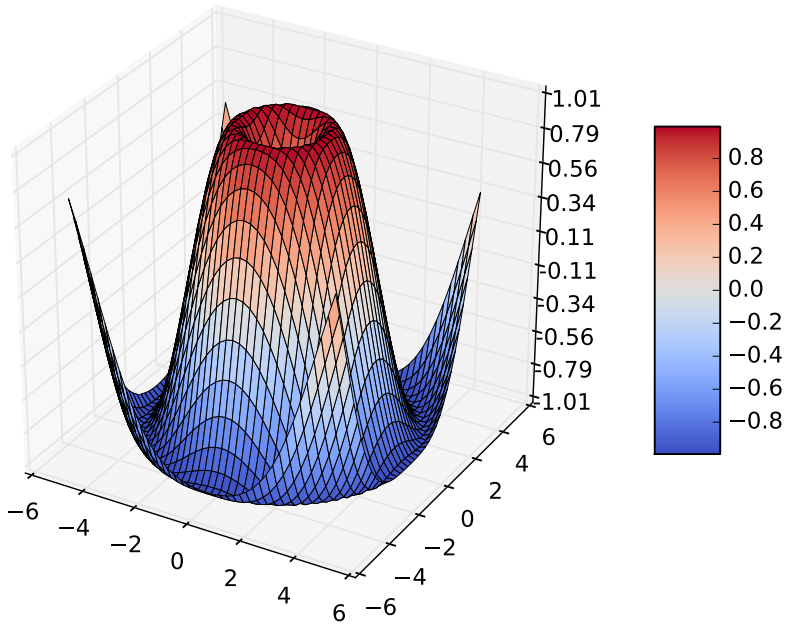
3D Plot

```
from mpl_toolkits.mplot3d import Axes3D;  import numpy as np
from matplotlib import cm; import matplotlib.pyplot as plt;
from matplotlib.ticker import LinearLocator,FormatStrFormatter

fig = plt.figure(); ax = fig.gca(projection='3d')
X = np.arange(-5, 5, 0.25); Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2); Z = np.sin(R)
surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
        cmap=cm.coolwarm,linewidth=0, antialiased=False)
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```

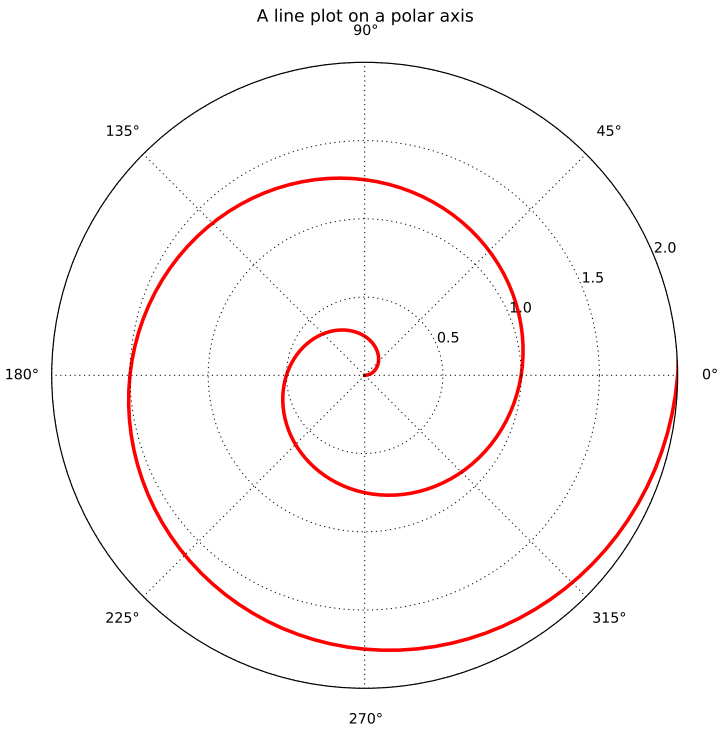

3D plot (Bild)



Polar Plot (Code)

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4
5  r = np.arange(0, 3.0, 0.01)
6  theta = 2 * np.pi * r
7
8  ax = plt.subplot(111, polar=True)
9  ax.plot(theta, r, color='r', linewidth=3)
10 ax.set_rmax(2.0)
11 ax.grid(True)
12
13 ax.set_title("A␣line␣plot␣on␣a␣polar␣axis", va='bottom')
14 plt.show()
```


Polar Plot (Bild)



Of course you can use TEX (code)

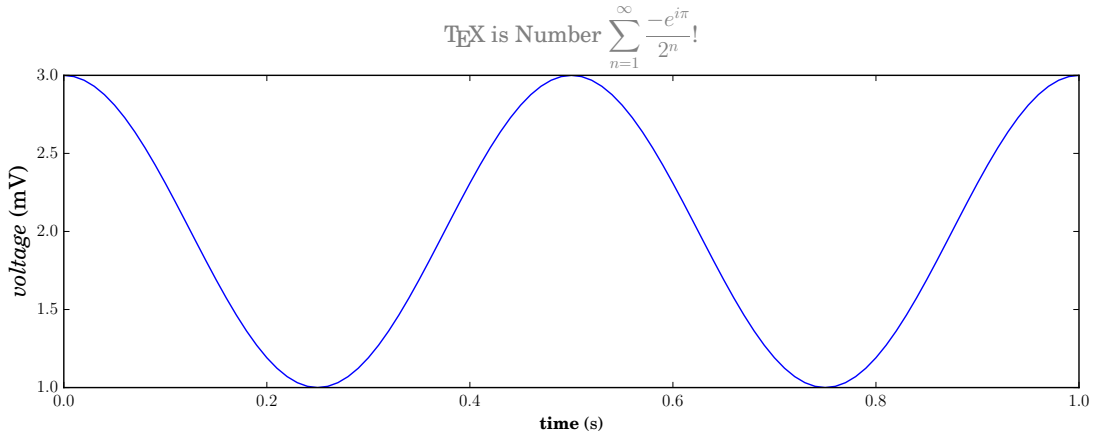
```
import numpy as np; import matplotlib.pyplot as plt
t = np.arange(0.0, 1.0 + 0.01, 0.01)
s = np.cos(4 * np.pi * t) + 2

plt.rc('text', usetex=True);  plt.rc('font', family='serif')
plt.plot(t, s)

plt.xlabel(r'\textbf{time} (s)')
plt.ylabel(r'\textit{voltage} (mV)',fontsize=16)
plt.title(r"\TeX\ is Number "
       r"$\displaystyle\sum_{n=1}^\infty\frac{-e^{i\pi}}{2^n}$!",
       fontsize=16, color='gray')
# Make room for the ridiculously large title.
plt.subplots_adjust(top=0.8)
plt.savefig('tex_demo')
plt.show()
```

It's all about that (drawing) space: *Matplotlib*

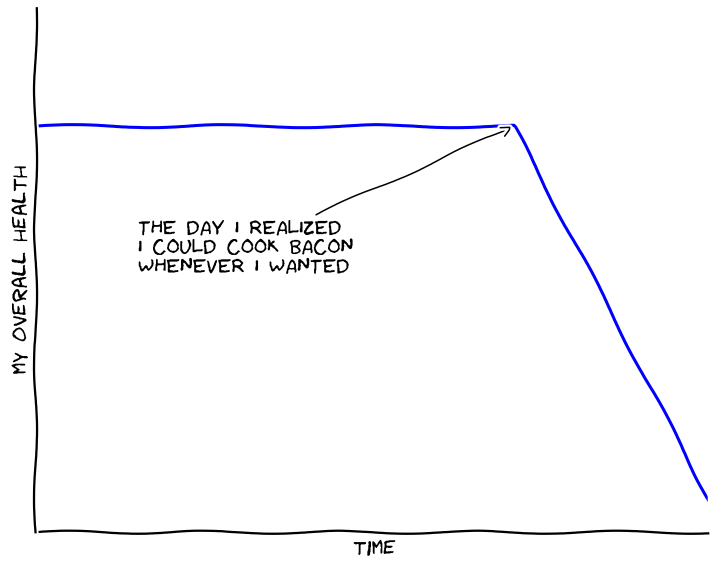
Of course you can use TEX (code)



Scientific and beyond: XKCD (code)

```
1  import matplotlib.pyplot as plt; import numpy as np
2
3  plt.xkcd()
4   # Based on 'Stove Ownership' from XKCD by Randall Monroe
5  fig = plt.figure();  ax = fig.add_axes((0.1, 0.2, 0.8, 0.7))
6  plt.xticks([]);  plt.yticks([]); ax.set_ylim([-30, 10])
7
8  data = np.ones(100); data[70:] -= np.arange(30)
9  plt.annotate(
10  'THE␣DAY␣I␣REALIZED\nI␣COULD␣COOK␣BACON\nWHENEVER␣I␣WANTED',
11   xy=(70, 1), arrowprops=dict(arrowstyle='->'),
12   xytext=(15, -10))
13
14 plt.plot(data);
15 plt.xlabel('time'); plt.ylabel('my␣overall␣health')
16 plt.show()
```


Scientific and beyond: XKCD (Bild)



In Debian muss das Paket: `fonts-humor-sans` installiert sein. Weiterhin muss der möglicherweise schon erstellte font-cache mittels `rm ./.cache/matplotlib/fontList.py3k.cache` gelöscht werden.