

# Einführungskurs Python

---

Dominik Dürschnabel

Mai 2019



# Wissenschaftliches Arbeiten mit Python

---

# Ziele des Abschnitts

Ziele

# Ziele des Abschnitts

## Ziele

- ▶ Vorstellung von „Standard-Bibliotheken“

# Ziele des Abschnitts

## Ziele

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

# 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:

# 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 Zusammenhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)

# 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 Zusammenhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- ▶ Lösen von **Optimierungsproblemen**.



# 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 Zusammenhängen (2D Plot, 3D Plot, Animationen, bis hin zu fotorealistischen Darstellungen)
- ▶ Lösen von **Optimierungsproblemen**.
- ▶ **Datenanalyse**, zum Beispiel mittels importierten Paketen von R.

# 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 Zusammenhängen (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**

# 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 Zusammenhängen (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**.

# 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 Zusammenhängen (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**.

# 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 Zusammenhängen (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?

---

<sup>1</sup>Basic Linear Algebra Subprograms

# Was ist NumPy?

- Spracherweiterung zu Python

---

<sup>1</sup>Basic Linear Algebra Subprograms



# Was ist NumPy?

- ▶ Spracherweiterung zu Python
- ▶ Anwendungsfeld ist (schnelles) numerisches Rechnen

---

<sup>1</sup>Basic Linear Algebra Subprograms

# Was ist NumPy?

- ▶ Spracherweiterung zu Python
- ▶ Anwendungsfeld ist (schnelles) numerisches Rechnen
- ▶ stellt mehrdimensionale Arrays und Operatoren dafür zur Verfügung

---

<sup>1</sup>Basic Linear Algebra Subprograms

# 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<sup>1</sup> und LAPACK<sup>2</sup> aus.

---

<sup>1</sup>Basic Linear Algebra Subprograms

# 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<sup>1</sup> und LAPACK<sup>2</sup> aus.

Woher bekomme ich es?

---

<sup>1</sup>Basic Linear Algebra Subprograms

# 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<sup>1</sup> und LAPACK<sup>2</sup> aus.

## Woher bekomme ich es?

- ▶ In Ubuntu OS: `sudo apt-get install python3-numpy`

---

<sup>1</sup>Basic Linear Algebra Subprograms

# 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<sup>1</sup> und LAPACK<sup>2</sup> aus.

## Woher bekomme ich es?

- ▶ In Ubuntu OS: `sudo apt-get install python3-numpy`
- ▶ Die neueste Version ist erhältlich unter:  
`git clone git://github.com/numpy/numpy.git` `numpy`

---

<sup>1</sup>Basic Linear Algebra Subprograms

# 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<sup>1</sup> und LAPACK<sup>2</sup> aus.

## Woher bekomme ich es?

- ▶ In Ubuntu OS: `sudo apt-get 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`

---

<sup>1</sup>Basic Linear Algebra Subprograms

# 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<sup>1</sup> und LAPACK<sup>2</sup> aus.

## Woher bekomme ich es?

- ▶ In Ubuntu OS: `sudo apt-get 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
```

---

<sup>1</sup>Basic Linear Algebra Subprograms



# Einschub Package-installer (pip)

# Einschub Package-installer (pip)

- ▶ Python besitzt seinen eigenen Package-installer pip.

# 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).

# 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

# 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)

# 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

# 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'`

# 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 apt-get 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)  # what is the difference between this...
6 print(M*M)  # ... and this?
7 print(M.I)  # matrices have inverse by default
8 v= np.matrix([[1],[2],[3]])
9 print(M*v)  # does what one expects
```

# 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)  # what is the difference between this...
6 print(M*M)  # ... and this?
7 print(M.I)  # matrices have inverse by default
8 v= np.matrix([[1],[2],[3]])
9 print(M*v)  # does what one expects
```



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))
```

# 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!

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

## Reshape!

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

# Wissenschaftliches Arbeiten mit Python

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

# matplotlib, pyplot and pylab?!?

- ▶ `matplotlib` ist das „whole package“



## matplotlib, pyplot and pylab?!?

- ▶ `matplotlib` ist das „whole package“
- ▶ `pyplot` ist ein Modul in `matplotlib`, vorzugsweise zu verwenden für nicht-interaktive Plots.

## matplotlib, pyplot and pylab?!?

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

# matplotlib, pyplot and pylab?!?

- ▶ matplotlib ist das „whole package“
- ▶ pyplot 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.

# matplotlib, pyplot and pylab?!?

- ▶ matplotlib ist das „whole package“
- ▶ pyplot 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.

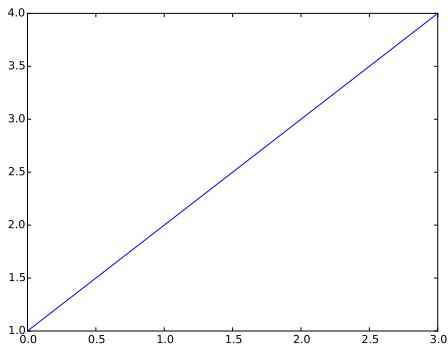
## Nutzung von matplotlib und Python3 unter Debian



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

## Simple plot (code)

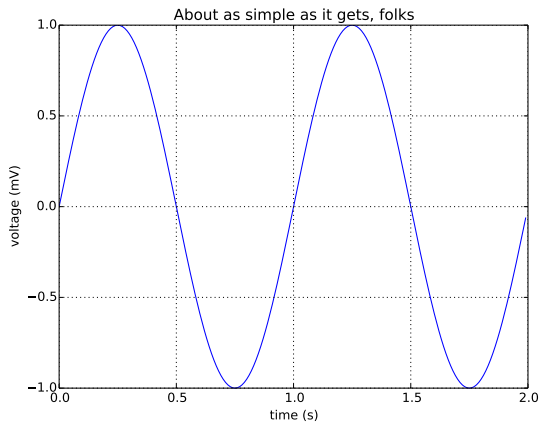
```
1 import matplotlib.pyplot as plt
2 plt.plot([1,2,3,4])
3 plt.show()
```



## 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)

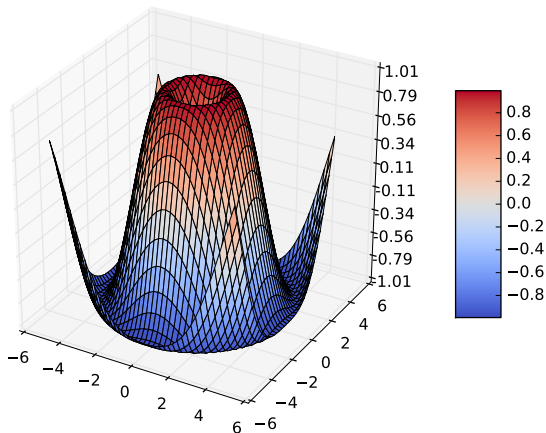


## 3D Plot

```
1 from mpl_toolkits.mplot3d import Axes3D; import numpy as np
2 from matplotlib import cm; import matplotlib.pyplot as plt;
3 from matplotlib.ticker import LinearLocator, FormatStrFormatter
4
5 fig = plt.figure(); ax = fig.gca(projection='3d')
6 X = np.arange(-5, 5, 0.25); Y = np.arange(-5, 5, 0.25)
7 X, Y = np.meshgrid(X, Y)
8 R = np.sqrt(X**2 + Y**2); Z = np.sin(R)
9 surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
10                        cmap=cm.coolwarm,linewidth=0, antialiased=False)
11 ax.set_zlim(-1.01, 1.01)
12 ax.zaxis.set_major_locator(LinearLocator(10))
13 ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
14
15 fig.colorbar(surf, shrink=0.5, aspect=5)
16 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)



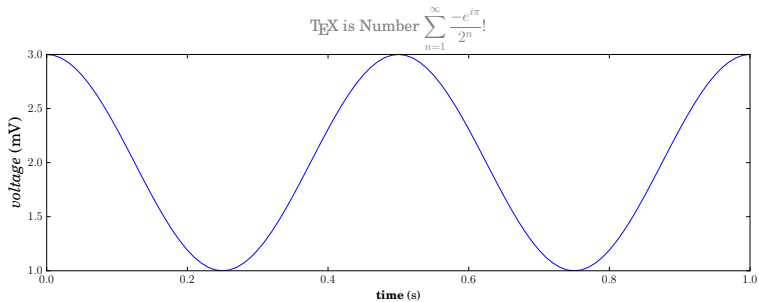
Of course you can use T<sub>E</sub>X (code)

```

1 import numpy as np; import matplotlib.pyplot as plt
2 t = np.arange(0.0, 1.0 + 0.01, 0.01)
3 s = np.cos(4 * np.pi * t) + 2
4
5 plt.rc('text', usetex=True); plt.rc('font', family='serif')
6 plt.plot(t, s)
7
8 plt.xlabel(r'\textbf{time}(s)')
9 plt.ylabel(r'\textit{voltage}(mV)', fontsize=16)
10 plt.title(r"\TeX\ is Number "
11           r"$\displaystyle\sum_{n=1}^{\infty}\frac{-e^{i\pi}}{2^n}$"
12           fontsize=16, color='gray')
13 # Make room for the ridiculously large title.
14 plt.subplots_adjust(top=0.8)
15 plt.savefig('tex_demo')
16 plt.show()

```

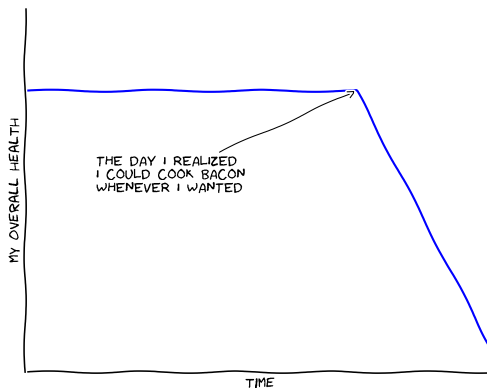
Of course you can use T<sub>E</sub>X (code)



## 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.

# Wissenschaftliches Arbeiten mit Python

Symbolisches Rechnen: SymPy



# Es geht eine Menge...

- ▶ Einfache Terme umformen via einfachen arithmetischen Operationen.

# Es geht eine Menge...

- ▶ Einfache Terme umformen via einfachen arithmetischen Operationen.
- ▶ Rechnen mit Polynomen (bis hin zu Gröbnerbasen).

# Es geht eine Menge...

- ▶ Einfache Terme umformen via einfachen arithmetischen Operationen.
- ▶ Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- ▶ Berechnen von Ableitungen, Integralen und Taylorreihen.

# Es geht eine Menge...

- ▶ Einfache Terme umformen via einfachen arithmetischen Operationen.
- ▶ Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- ▶ Berechnen von Ableitungen, Integralen und Taylorreihen.
- ▶ Symbolisches lösen von Gleichungen und Differentialgleichungen.

# Es geht eine Menge...

- ▶ Einfache Terme umformen via einfachen arithmetischen Operationen.
- ▶ Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- ▶ Berechnen von Ableitungen, Integralen und Taylorreihen.
- ▶ Symbolisches lösen von Gleichungen und Differentialgleichungen.
- ▶ Symbolisches Rechnen mit Matrizen.

# Es geht eine Menge...

- ▶ Einfache Terme umformen via einfachen arithmetischen Operationen.
- ▶ Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- ▶ Berechnen von Ableitungen, Integralen und Taylorreihen.
- ▶ Symbolisches lösen von Gleichungen und Differentialgleichungen.
- ▶ Symbolisches Rechnen mit Matrizen.
- ▶ Rechnen mit Punkten, Schnitten und Tangenten

# Es geht eine Menge...

- ▶ Einfache Terme umformen via einfachen arithmetischen Operationen.
- ▶ Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- ▶ Berechnen von Ableitungen, Integralen und Taylorreihen.
- ▶ Symbolisches lösen von Gleichungen und Differentialgleichungen.
- ▶ Symbolisches Rechnen mit Matrizen.
- ▶ Rechnen mit Punkten, Schnitten und Tangenten
- ▶ Nutzen von physikalische Einheiten.

# Es geht eine Menge...

- ▶ Einfache Terme umformen via einfachen arithmetischen Operationen.
- ▶ Rechnen mit Polynomen (bis hin zu Gröbnerbasen).
- ▶ Berechnen von Ableitungen, Integralen und Taylorreihen.
- ▶ Symbolisches lösen von Gleichungen und Differentialgleichungen.
- ▶ Symbolisches Rechnen mit Matrizen.
- ▶ Rechnen mit Punkten, Schnitten und Tangenten
- ▶ Nutzen von physikalische Einheiten.

## Woher bekommt man es?

- ▶ `sudo apt-get install python-sympy (!Python2 only!)`, oder
- ▶ `pip3 install --user sympy`



# Los geht's SymPy: Klassifizieren und in chic

```
1 import sympy as sy
2 x = sy.symbols('x') # Klassifiziert x zum ein Symbol
3 y, z = sy.symbols('y_ z')
4 print(x+y*z)
5 sy.pprint(x+y*z) # PrettyPrint-Funktion von sympy
6
7 f = sy.Function('f')
8 sy.pprint(sy.Integral(f(x), x))
9
10 f = 1/sy.cos(x)
11 sy.pprint(f.series(x, 0, 10))
12
13 sy.pprint(sy.expand((x+y)**5))
```

# SymPy: Funktionen, Grenzwerte und Integrale

```
1 from sympy import *
2 init_printing() #Schaltet pretty printing generell an
3 x = Symbol('x')
4 f = Function('f'); f = sqrt(x**3 - 5*x + 2)
5 limit(f, x, oo); limit(f, x, -oo); limit(f, x, 0)
6
7 g = Function('g'); g = (x+1) / (x-1)
8 limit(g, x, oo); limit(g, x, -oo); limit(g, x, 0)
9
10 limit(f/g,x,0); limit(f/g,x,1)
```

# SymPy: Simplify and calculus

```
1 from sympy import *
2 init_printing()
3 x, y, z = symbols('x_y_z')
4
5 simplify(sin(x)**2 + cos(x)**2)
6 simplify((x**3 + x**2 - x - 1)/(x**2 + 2*x + 1))
7 simplify(gamma(x)/gamma(x - 2))
```

```
1 diff(x**4, x, 2); diff(x**4, x, 3)
2 diff(exp((x*y*z + y)**2), x)
3 integrate(cos(x), x)
4 integrate(exp(-x), (x, 0, oo))
5 integrate(exp(-x**2 - y**2), (x, -oo, oo), (y, -oo, oo))
```

# SymPy: Solve my Equation!

Gleichungen werden mit `Eq(linker Term, rechter Term)` definiert.

```
1 from sympy import *
2 init_printing()
3 x, y, z = symbols('x y z')
4 f = Function('f')
5
6 solve(Eq(4, x*2), x)
7
8 solve(Eq(y*x-4*x*z, z**2-4*x), x)
```

```
1 f = Function('f')
2 k = symbols('k')
3 g = f(x).diff(x, x) + k**2*f(x)
4 diffeq = Eq(g, 0)
5 dsolve(diffeq, f(x))
```

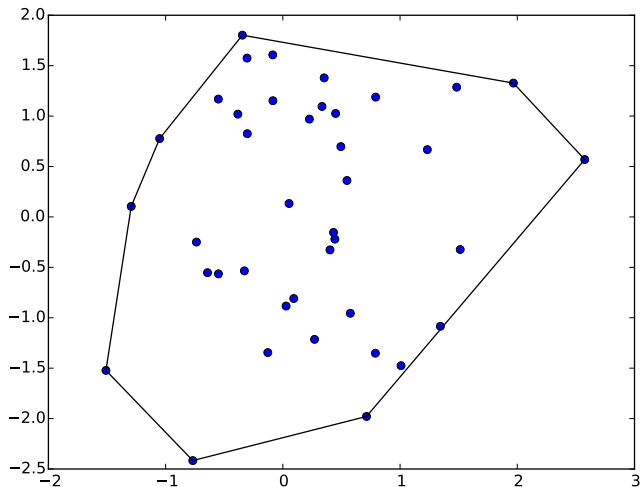
# Wissenschaftliches Arbeiten mit Python

SciPy und Freunde, was geht noch?

## Konvexen Hülle einer Punktmenge (scipy.spatial)

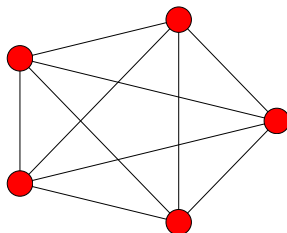
```
1 from scipy.spatial import ConvexHull
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 points = np.random.randn(40, 2)
6 hull = ConvexHull(points)
7 plt.plot(points[:,0], points[:,1], 'o')
8 for simplex in hull.simplices:
9     plt.plot(points[simplex,0], points[simplex,1], 'k-')
10
11 plt.savefig('con.pdf')
12 plt.show()
```

# Konvexen Hülle einer Punktmenge (scipy.spatial) (bild)



# Graphs Graphs Graphs (networkx)

```
1 import networkx as nx
2 import matplotlib.pyplot as plt; import pylab
3
4 G = nx.complete_graph(5)
5 nx.draw_circular(G,node_size=1500)
6 pylab.show()
```





## Strongly connected components (networkx)

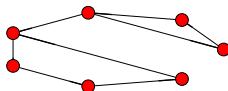
```
pip3 install --user networkx
```

```
1 import networkx as nx
2 import matplotlib.pyplot as plt; import pylab
3 G = nx.DiGraph()
4 for e in [(1,2),(2,3),(3,1),(3,4),(4,5),(5,6),(6,7),(7,4)]:
5     G.add_edge(*e)
6
7 SCC = nx.strongly_connected_components(G)
8 SCCG = nx.strongly_connected_component_subgraphs(G)
9 for c in SCC:
10     print(c)
11
12 for g in SCCG:
13     nx.draw_circular(g,node_size=1500)
14     pylab.show()
```

# Strongly connected components (networkx)

```
pip3 install --user networkx
```

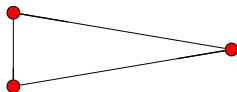
```
1 import networkx as nx
2 import matplotlib.pyplot as plt; import pylab
3 G = nx.DiGraph()
4 for e in [(1,2),(2,3),(3,1),(3,4),(4,5),(5,6),(6,7),(7,4)]:
5     G.add_edge(*e)
6
7 SCC = nx.strongly_connected_components(G)
8 SCCG = nx.strongly_connected_component_subgraphs(G)
9 for c in SCC:
10     print(c)
11
12 for g in SCCG:
13     nx.draw_circular(g,node_size=1500)
14     pylab.show()
```



# Strongly connected components (networkx)

```
pip3 install --user networkx
```

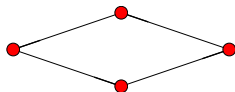
```
1 import networkx as nx
2 import matplotlib.pyplot as plt; import pylab
3 G = nx.DiGraph()
4 for e in [(1,2),(2,3),(3,1),(3,4),(4,5),(5,6),(6,7),(7,4)]:
5     G.add_edge(*e)
6
7 SCC = nx.strongly_connected_components(G)
8 SCCG = nx.strongly_connected_component_subgraphs(G)
9 for c in SCC:
10     print(c)
11
12 for g in SCCG:
13     nx.draw_circular(g,node_size=1500)
14     pylab.show()
```



# Strongly connected components (networkx)

```
pip3 install --user networkx
```

```
1 import networkx as nx
2 import matplotlib.pyplot as plt; import pylab
3 G = nx.DiGraph()
4 for e in [(1,2),(2,3),(3,1),(3,4),(4,5),(5,6),(6,7),(7,4)]:
5     G.add_edge(*e)
6
7 SCC = nx.strongly_connected_components(G)
8 SCCG = nx.strongly_connected_component_subgraphs(G)
9 for c in SCC:
10     print(c)
11
12 for g in SCCG:
13     nx.draw_circular(g,node_size=1500)
14     pylab.show()
```

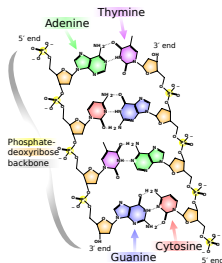


# Lifefoms, (scikits.bio)

```
pip3 install --user scikit-bio
```

Ein Problem aus der Biologie: Finde die längste Folge von Purinen (Guanin oder Adenin) in einem gegebenen DNS-Strang.

```
1 from skbio import NucleotideSequence
2 nuc = NucleotideSequence.read(
3     'single_sequence1.fasta', seq_num=1)
4
5 max(nuc.find_features('purine_run'), key=lambda e: len(e[2]))
```



# Making things faster: Numba

„Numba works by generating optimized machine code using the LLVM compiler [...] and is designed to integrate with the Python scientific software stack.“

```
1 from numba import jit; from numpy import arange
2 # jit decorator tells Numba to compile this function.
3 @jit
4 def sum2d(arr):
5     M, N = arr.shape; result = 0.0
6     for i in range(M):
7         for j in range(N):
8             result += arr[i,j]
9     return result
10
11 a = arange(9).reshape(3,3)
12 print(sum2d(a))
```

# NLTK - Natural Language Toolkit

„NLTK is a leading platform for building Python programs to work with human language data.“

```
1 import nltk
2 s = """The lecture by Tom about Python was fantastic."""
3 tokens = nltk.word_tokenize(s)
4 ['The', 'lecture', 'by', 'Tom', 'about',
5  'Python', 'was', 'fantastic', '.']
6 tagged = nltk.pos_tag(tokens)
7 [('The', 'DT'), ('lecture', 'NN'), ('by', 'IN'),
8  ('Tom', 'NNP'), ('about', 'IN'), ('Python', 'NNP'),
9  ('was', 'VBD'), ('fantastic', 'JJ'), ('.', '.')]

```

Wobei: DT - Determiner, NN - Noun, singular or mass,  
IN - Preposition or subordinating conjunction  
NNP - Proper noun, singular, VBD - Verb, past tense

# Pandas - data manipulation and analysis

## Beispiel Pandas Docu

```
1 import pandas as pd #this is how I usually import pandas
2 names = ['Bob','Jessica','Mary','John','Mel']
3 births = [968, 155, 77, 578, 973]
4 BabyDataSet = list(zip(names,births))
5 [('Bob', 968), ('Jessica', 155), ('Mary', 77),
6  ('John', 578), ('Mel', 973)]
7 pd.DataFrame(data = BabyDataSet, columns=['Names', 'Births'])
```

|   | Names   | Births |
|---|---------|--------|
| 0 | Bob     | 968    |
| 1 | Jessica | 155    |
| 2 | Mary    | 77     |
| 3 | John    | 578    |
| 4 | Mel     | 973    |



## Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen

## Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen
- ▶ ScientificPython, theoretische Biophysik Simulation und mehr

# Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen
- ▶ ScientificPython, theoretische Biophysik Simulation und mehr
- ▶ Thuban, geographische Datenanalyse

# Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen
- ▶ ScientificPython, theoretische Biophysik Simulation und mehr
- ▶ Thuban, geographische Datenanalyse
- ▶ TensorFlow

# Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen
- ▶ ScientificPython, theoretische Biophysik Simulation und mehr
- ▶ Thuban, geographische Datenanalyse
- ▶ TensorFlow
- ▶ Plotly

# Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen
- ▶ ScientificPython, theoretische Biophysik Simulation und mehr
- ▶ Thuban, geographische Datenanalyse
- ▶ TensorFlow
- ▶ Plotly
- ▶ SciKit-Learn

# Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen
- ▶ ScientificPython, theoretische Biophysik Simulation und mehr
- ▶ Thuban, geographische Datenanalyse
- ▶ TensorFlow
- ▶ Plotly
- ▶ SciKit-Learn
- ▶ Theano

# Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen
- ▶ ScientificPython, theoretische Biophysik Simulation und mehr
- ▶ Thuban, geographische Datenanalyse
- ▶ TensorFlow
- ▶ Plotly
- ▶ SciKit-Learn
- ▶ Theano
- ▶ Scrapy



# Tolle Pakete für die keine Zeit war.

- ▶ PsychoPy, cognitive science und neuroscience Experimente durchführen
- ▶ ScientificPython, theoretische Biophysik Simulation und mehr
- ▶ Thuban, geographische Datenanalyse
- ▶ TensorFlow
- ▶ Plotly
- ▶ SciKit-Learn
- ▶ Theano
- ▶ Scrapy
- ▶ ...

# Wissenschaftliches Arbeiten mit Python

Sage

## Let's have some sage tea

Sage ist ein Konglomerat von Programmen, die jeweils für sich für spezielle Problemstellungen entwickelt wurden, vereint in einem Interface.

|                 |                    |
|-----------------|--------------------|
| Algebra         | Singular, PolyBoRi |
| Analysis        | Maxima, SymPy      |
| Zahlentheorie   | PARI/GP, NTL       |
| Numerik         | NumPy, SciPy       |
| Statistik       | R                  |
| Lineare Algebra | LinBox, LAPACK     |
| Graphentheorie  | NetworkX           |
| Gruppentheorie  | GAP                |

Wie kann man es benutzen?

## Let's have some sage tea

Sage ist ein Konglomerat von Programmen, die jeweils für sich für spezielle Problemstellungen entwickelt wurden, vereint in einem Interface.

|                 |                    |
|-----------------|--------------------|
| Algebra         | Singular, PolyBoRi |
| Analysis        | Maxima, SymPy      |
| Zahlentheorie   | PARI/GP, NTL       |
| Numerik         | NumPy, SciPy       |
| Statistik       | R                  |
| Lineare Algebra | LinBox, LAPACK     |
| Graphentheorie  | NetworkX           |
| Gruppentheorie  | GAP                |

Wie kann man es benutzen?

- ▶ Mit einem browser, z. B. via <https://cloud.sagemath.com/>

# Let's have some sage tea

Sage ist ein Konglomerat von Programmen, die jeweils für sich für spezielle Problemstellungen entwickelt wurden, vereint in einem Interface.

|                 |                    |
|-----------------|--------------------|
| Algebra         | Singular, PolyBoRi |
| Analysis        | Maxima, SymPy      |
| Zahlentheorie   | PARI/GP, NTL       |
| Numerik         | NumPy, SciPy       |
| Statistik       | R                  |
| Lineare Algebra | LinBox, LAPACK     |
| Graphentheorie  | NetworkX           |
| Gruppentheorie  | GAP                |

## Wie kann man es benutzen?

- ▶ Mit einem browser, z. B. via <https://cloud.sagemath.com/>
- ▶ Lokal installiert und mittels IPython basierter Konsole.

## Let's have some sage tea

Sage ist ein Konglomerat von Programmen, die jeweils für sich für spezielle Problemstellungen entwickelt wurden, vereint in einem Interface.

|                 |                    |
|-----------------|--------------------|
| Algebra         | Singular, PolyBoRi |
| Analysis        | Maxima, SymPy      |
| Zahlentheorie   | PARI/GP, NTL       |
| Numerik         | NumPy, SciPy       |
| Statistik       | R                  |
| Lineare Algebra | LinBox, LAPACK     |
| Graphentheorie  | NetworkX           |
| Gruppentheorie  | GAP                |

### Wie kann man es benutzen?

- ▶ Mit einem browser, z. B. via <https://cloud.sagemath.com/>
- ▶ Lokal installiert und mittels IPython basierter Konsole.
- ▶ Via Python durch Nutzung der Sage-Bibliotheken.

# Wissenschaftliches Arbeiten mit Python

Final Chapter

## Lift off to „Jupyter“

„ALL THESE WORLDS ARE YOURS - EXCEPT EUROPA.  
ATTEMPT NO LANDINGS THERE.“

-- HAL

Now: Life demonstration of Jupyter.