(Scientific) Python und Jupyter Notebooks in der Hochschullehre

Wie profitieren Schüler / Studierende von der Programmiersprache Python?

- Python im Vergleich zu anderen Programmiersprachen
- Python auf einem Rechner verwenden (Distributionen, IDEs)
- Interaktives Software Prototyping mit IPython
- Web-basiertes Arbeiten mit Jupyter Notebooks
- Wissenschaftliches Rechnen mit Scientific Python: Matrixalgebra, symbolische Mathematik, Datenvisualisierung, animierte Grafiken, Bildverarbeitung, Messautomatisierung, Audiosignalerzeugung, numerische Mathematik, Signalverarbeitung, Simulationen.
- Physical Computing mit Python auf Einplatinencomputern
- Micropython auf dem ESP32 Mikrocontroller

Zu meiner Person

- Hatte exzellenten Mathe/Physiklehrer
- Physikstudium an Uni Heidelberg / U of Oregon, USA.

► Promotion Max Planck Institut Halle (Saale) + Bosch in Reutlingen
» MEMS, Oberflächenphysik

Bildquelle: www.bosch.de

► Carl Zeiss in Jena
» Technische Optik

Bildquelle: www.zeiss.de

► Sick AG bei Freiburg
» Optosensorik (Lidar), Bildverarbeitung

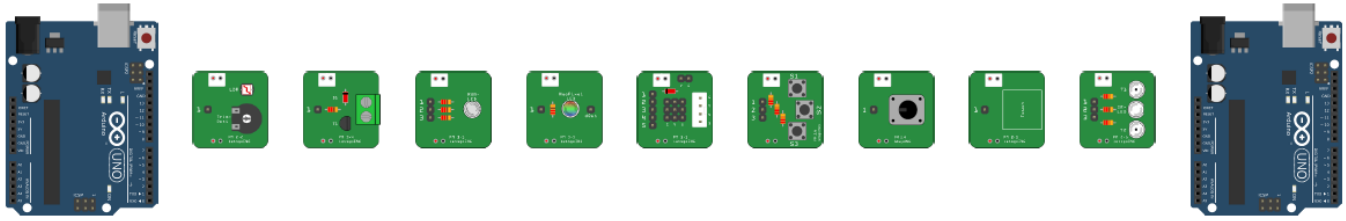
Bildquelle: www.sick.de

► Seit 2011 Fachhochschule Reutlingen, Professur Mess- und Sensortechnik Fakultät Technik, Studienbereich Mechatronik.
Begeistert von *Arduino, RaspberryPi/BeagleBone* und *Python.*

Bildungssystem letsgoING an der Hochschule RT

letsgoING

Mikrocontroller macht Schule



Grundkonzepte letsgoING bezogen auf Informatikanteil:

Konzentration auf das Wesentliche mit grafischer Programmierung
» Erlernen der Logik und Denkweise beim Programmieren.

Übersichtliche Programmierumgebung
» Funktionen sind logisch sortiert und durch Farbe und Form erkennbar.

Skalierbare Programmierumgebung
» Umstieg auf textbasierte Programmierung einfach möglich, C-Code wird erzeugt.

Python: Sehr ähnliches Leitbild!

Ursprung der Programmiersprache Python

Bildquelle: wikipedia.de

Erscheinungsjahr:	1991, akt. Version 3.8
Designer:	G. van Rossum (Holland!)
Sprachenart:	Hochsprache, Interpreter
Paradigmen:	multiparadigmisch
Betriebssystem:	plattformunabhängig
Lizenz:	Open Source

Bildquelle: mdr.de/kultur/erste-folge-monty-python-ausgestrahlt-100.html

„Python“ angelehnt an „Monty Python“

Englische Komikergruppe,
5.10.1969 erste Sendung in der BBC.
Kernkompetenz: Konventionen und Tabus brechen.

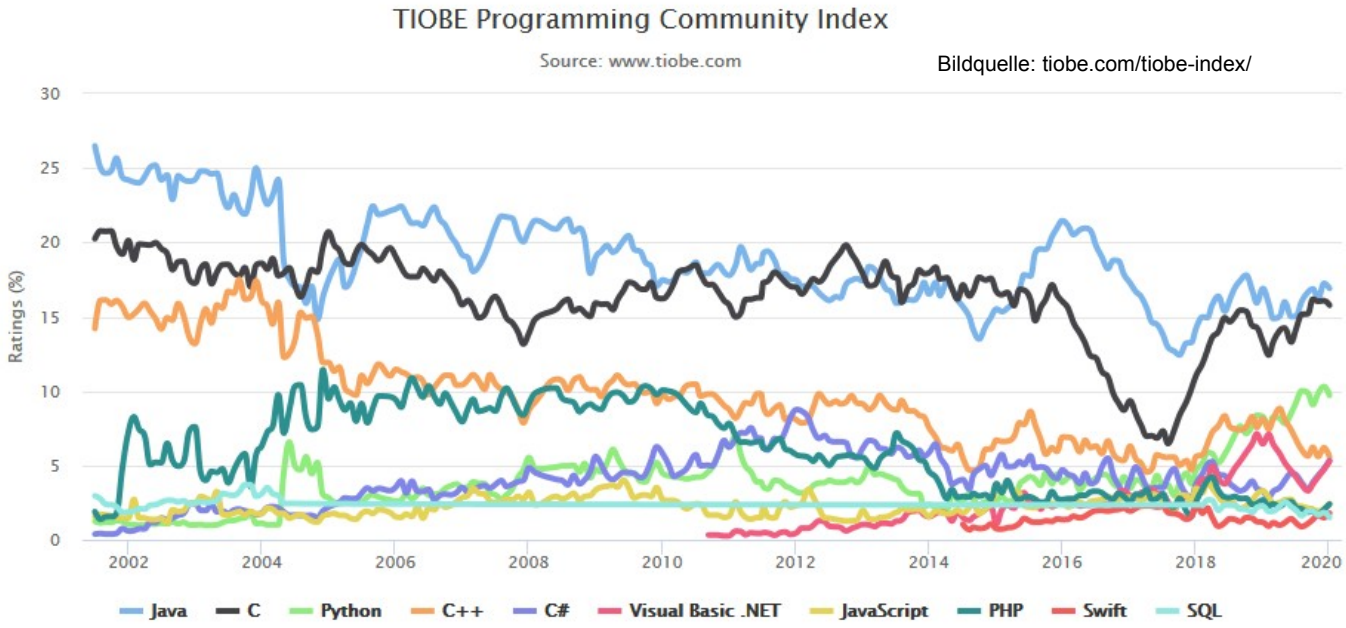
... in Bezug auf Programmiersprache z.B.:
» Strukturierung durch Einrückungen.
» Iterieren über Listenelemente.
» Dynamisches Ändern Variablentyp.

Welche ist die beste Programmiersprache?

Hängt davon ab,

- für welchen Zweck?
  - z.B. Anwendersoftware
  - z.B. Lehre
  - z.B. Embedded

- für welche Zielgruppe?
  - z.B. Schüler / Studierende
  - z.B. KI-Wissenschaftler



Bildquelle: tiobe.com/tiobe-index/

Trend Rapid Prototyping: Arduino, Raspberry Pi, 3D-Drucker, Simulationswerkzeuge

Python = erfolgreiche Rapid Prototyping Programmiersprache
da inzwischen oft Kosten Softwareentwicklung > Kosten Rechenleistung

Tipp: Geschichtliche Entwicklung der Programmiersprachen:
» Podcast „Geschichte(n) der Programmiersprachen in einer Langen Nacht“, Dlf vom 7.9.2019.
`deutschlandfunk.de/geschichte-n-der-programmiersprachen-in-einer-langen-nacht.704.de.html?dram:article_id=454167`

Wieso nicht Fokus auf Programmiersprache C?

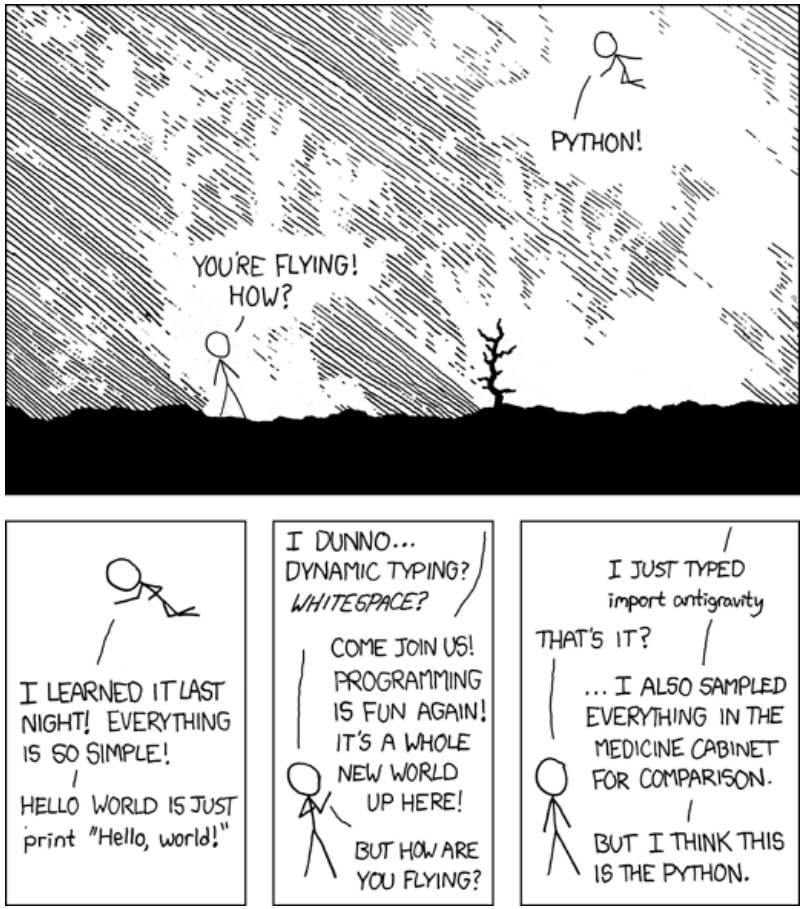
oder „Latein“?

C = „Englisch“ der Programmiersprachen: universell, etabliert, performant.

Motivation Einsatz Python an Schulen und Hochschulen:

- Flache Lernkurve bei Python, darauf aufbauend dann C/C++ lernen.
  z.B. letsgoING: Erst ArduBlock dann C-Code-Generator.
- Python als Interpretersprache optimal für exploratives Arbeiten und Rapid Prototyping.
- Prozedural, objektorientiert oder funktional verwendbar.
- „*Batteries Included*“ = Viele von Bibliotheken inklusive.
- Einfache Syntax, fremder Code gut verständlich.
- Open Source, kostenlos.
- Viele Best Practice Beispiele

Bildquelle: xkcd.com/353/



Python als Open Source Alternative zu MATLAB

Python strukturell ähnlich zu MATLAB: Interpreter mit vorkompilierten C-/Fortran-Bibliotheken.
Umstieg von MATLAB einfach da Bibliotheken mit bewusst ähnlicher Syntax.

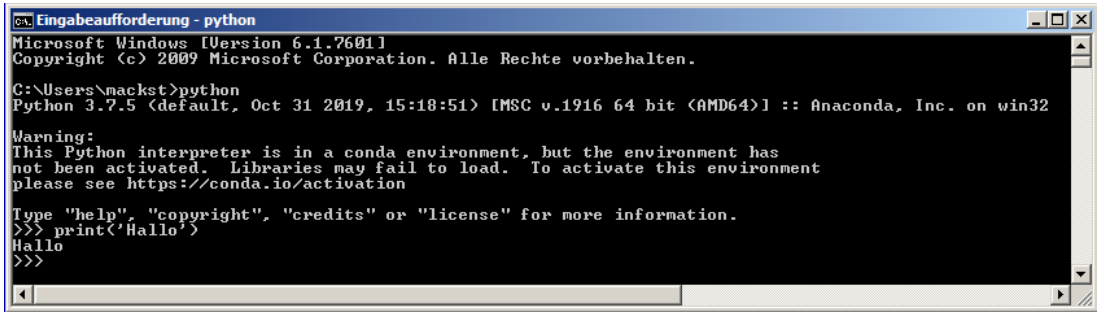
MATLAB » **teuer und proprietär** Python / Octave » **kostenlos und Open Source.**

- Rechtssicherheit: Keine Angst vor Linzenzverstößen z.B. bei Drittmittelforschungsprojekten, Python-Code auch für kommerzielle Zwecke.
- Keine Salami-Taktik: Zusätzliche Softwaremodule bei MATLAB nur gegen weitere Zahlung. MATLAB Toolboxen nicht für jede Plattform verfügbar.
- Investitionssicherheit: Unabhängig von Lizenzpolitik oder inhaltliche Ausrichtung des Software-Konzerns.
- Über Stack Overflow, Tutorials, Videos inzwischen weitaus mehr und bessere Hilfe als bei MATLAB (Crowd Intelligence).
- Python richtige Programmiersprache im Gegensatz zu MATLAB/Octave.
- Mitmachen bei Python (z.B. eigene Bibliothek veröffentlichen) motiviert.
- Python ist wichtigste Programmiersprache für künstliche Intelligenz.

Python auf dem Rechner verwenden

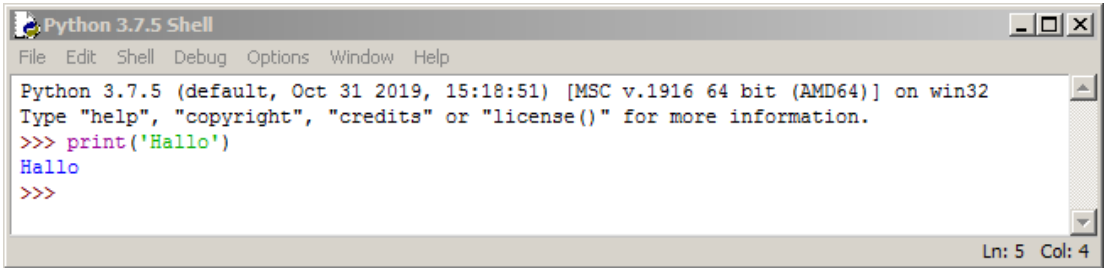
Rechnerplattform (Windows, iOS, Linux): Egal.

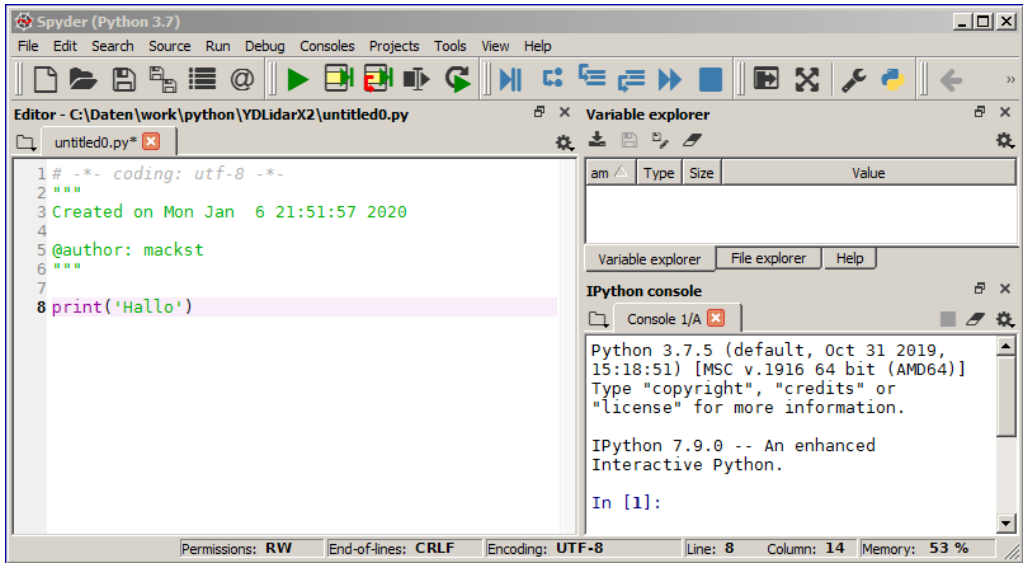
- Python aus der Kommandozeile:
  » Didaktisch gar nicht unklug,
  siehe Informatik-Buch Häberlein.



- Python über eine IDE:
  » „Integrierte Entwicklungsumgebung“ mit Editor, Syntax-Highlighting, Code Analysis, Debugger, integrierter Hilfe.
  Oft IDE inklusive: Idle, Thonny (RasPi).

  Für wissenschaftliches Rechnen:
  Spyder, PyCharm (optimal) oder Atom, Eclipse.





Python auf dem Rechner verwenden

Python 2 oder 3? Klare Antwort: **Nur !! Python 3.**

Falls kein Python inkl. IDE schon auf PC: Python-Distribution installieren!

Distributionen erleichtern Nachinstallation von IDEs + Bibliotheken und Updates erheblich!

Nicht-kommerzielle Distributionen für wissenschaftliches Rechnen:

Bildquelle: anaconda.com/distribution/

- Miniconda / Anaconda:
  Miniconda = Teilinstallation Anaconda

- WinPython:
  Nur Windows, auch als portable App,
  sonst ähnlich zu Anaconda

Bildquelle: winpython.github.io/

Ansonsten Python-eigene Paketverwaltung „pip“ zur Installation von Bibliotheken / Erweiterungen.

Beispiel IDE Spyder mit IPython

Prinzip REPL „**Read - Eval - Print - Loop**“:

Iterativ, explorativ und interaktiv Code entwickeln mit der Kommandozeile (Python-Interpreter) oder besser noch mit **IPython** (=besonders komfortable Kommandozeile).

Beispiel:
String „Hallo Welt“ in Großbuchstaben überführen und am Leerzeichen teilen.

Textvervollständigung oder Abfrage Methoden/Attribute Objekt über TAB-Taste.

IPython bietet auch: Debugger, Datei-manager, Laufzeitmessung, ...

```
In [1]: txt='Hallo Welt'

In [2]: type(txt)
Out[2]: str

In [3]: str.upper?
Signature: str.upper(self, /)
Docstring: Return a copy of the string converted to uppercase.
Type:      method_descriptor

In [4]: txt_gross=txt.upper()

In [5]: txt_gross
Out[5]: 'HALLO WELT'

In [6]: str.split?
Signature: str.split(self, /, sep=None, maxsplit=-1)
Docstring:
Return a list of the words in the string, using sep as the delimiter string.

sep
  The delimiter according which to split the string.
  None (the default value) means split according to any whitespace,
  and discard empty strings from the result.
maxsplit
  Maximum number of splits to do.
  -1 (the default value) means no limit.
Type:      method_descriptor

In [7]: txt_gross.split()
Out[7]: ['HALLO', 'WELT']
```

Beispiel IDE Spyder mit IPython

Beispiel:
Verwendung Modul `math`, Mathematikfunktionen

```
In [1]: import math

In [2]: math.sin?
Signature: math.sin(x, /)
Docstring: Return the sine of x (measured in radians).
Type:      builtin_function_or_method

In [3]: math.pi?
Type:        float
String form: 3.141592653589793
Docstring:   Convert a string or number to a floating point number, if possible.

In [4]: math.sin(3.4*math.pi)
Out[4]: -0.9510565162951534

In [5]: Out[4]**4
Out[5]: 0.8181356214843416

In [6]: Out[5]-Out[4]
Out[6]: 1.769192137779495
```

Anders als textbasierte Kommandozeilen: IPython beherrscht Grafikdarstellung z.B. für Plots, Bilder, symbolische Mathematik oder mathematische Formeln (TeX).

```
In [1]: import sympy

In [2]: sympy.init_printing()

In [3]: x = sympy.Symbol("x")

In [4]: sympy.sqrt(x ** 2)
Out[4]:
```

$$\sqrt{x^2}$$

```
In [1]: from IPython.display import Math

In [2]: Math(r'\frac{d}{dt}')
Out[2]:
```

$$\frac{d}{dt}$$

Jupyter Notebooks

Sowohl für Lernende als auch Lehrende wichtig:
» **Fremden Code verstehen, verwenden und weiterentwickeln.**

Voraussetzung hierfür an Fremdcode:

- Code gut strukturiert (grafisch formatiert) und dadurch lesbar.
- Code gut dokumentiert über Kommentare.
- Code über GitHub oder ähnliche Plattform versioniert als Package verfügbar.
- Code plattformunabhängig.

» Python erfüllt Voraussetzungen.

Jupyter Notebooks bieten noch viel mehr:

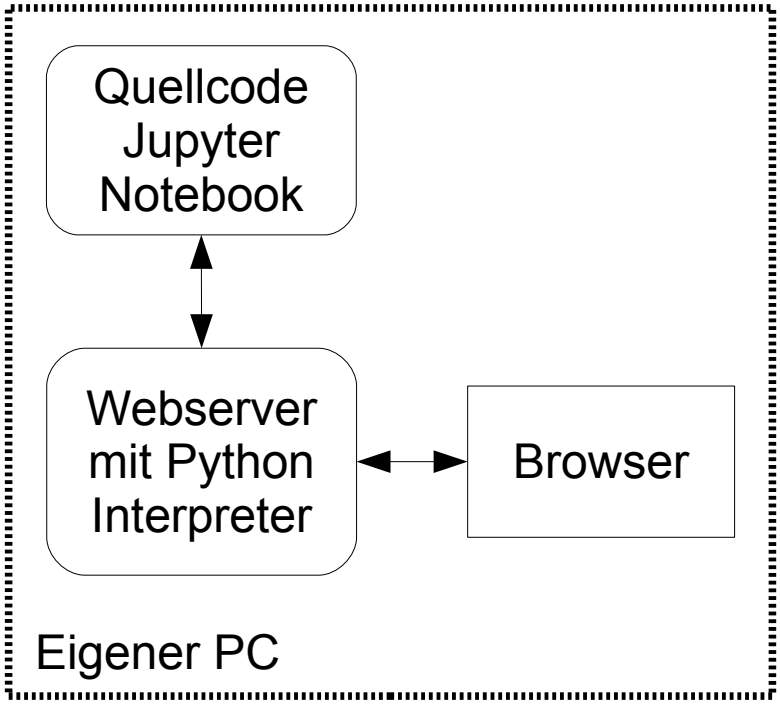
- Zugriff auf Fremdcode über Webbrowser, Ändern und Ausführen in Webbrowser.
- Programmoutput (inkl. Grafiken, Animationen) im Browser sichtbar.
- Zwischen Codezeilen Kommentare inkl. Grafiken wie Plots oder Formeln.
- Browserinhalte einfach in Formate wie PDF, HTML, TeX-Datei konvertierbar.

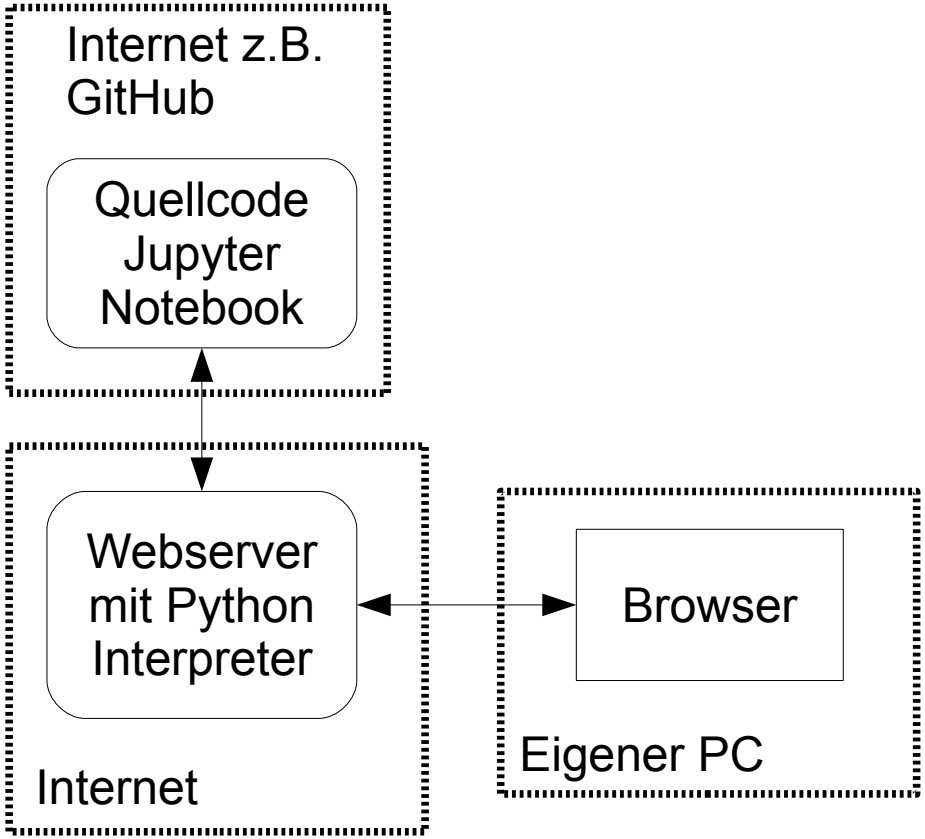
Viele Python-Lehrbücher als Jupyter Notebook frei verfügbar: » „Lebendiges“ Lehrbuch.
Jupyter Notebooks optimal für Doku selbst geschriebenen Codes.

Jupyter Notebooks

Jupyter Notebooks als *Webservice* von *Webserver* bereit gestellt. Webserver kann lokal auf selben Rechner sein, auf einem RasPi, irgendwo im Hochschulnetz oder im Internet.

Notebook im Netzwerk: Zugriff egal mit welchem Browser, egal womit (Smartphone, PC, Notebook), kann herunter geladen, (in IDE) lokal weiterentwickelt werden.





Jupyter Notebooks

Software für Erstellen und lokale Entwicklung von Jupyter Notebooks in Distributionen Anaconda / WinPython enthalten.

jupyter DglNumLoes Last Checkpoint: 12.09.2019 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown

Verwendung des Solvers `solve_ivp` aus `scipy.integrate` für eine GDGL ersten Grades

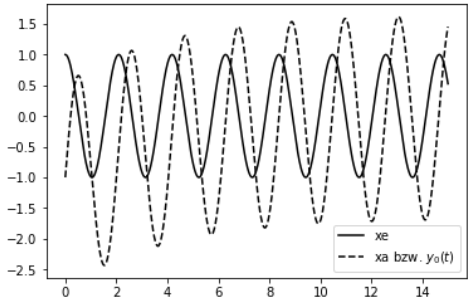
Im folgenden Beispielcode wird die GDGL

$$\frac{d}{dt}x_a = cx_e(t) - x_a/4,\ x_e(t) = \cos(3t) \qquad \frac{d}{dt}y_0 = c\cos(3t) - y_0/4$$

numerisch für $c = 5$ gelöst.

In [3]:
```
from scipy.integrate import solve_ivp

def xe(t): # input signal function of time, cosine oscillation
    return np.cos(3 * t)

def func_f(y, t, c): # function f() of ODE
    dydt = c * xe(t) - y / 4
    return dydt

tspan = np.linspace(0, 15, 1000) # sampling time values
y_init = [-1] # initial value of y

# numerical solving of ODE, solve_ivp() only accepts f(y,t) hence lambda function work around
sol = solve_ivp(lambda t, y: func_f(y, t, c=5), [tspan[0], tspan[-1]], y_init, t_eval=tspan)

#------------------------------------------------------------------------------
#----plotting results
#------------------------------------------------------------------------------
x_vals = np.zeros(np.size(sol.t))
for index,time in enumerate(sol.t):
    x_vals[index] = xe(time)
fig, ax = plt.subplots(num=1)
ax.plot(sol.t, xe(sol.t), 'k-', label='xe')
ax.plot(sol.t, sol.y[0,:], 'k--', label='xa bzw. $y_0(t)$')
ax.legend(loc='best')
```

Out[3]:
```
<matplotlib.legend.Legend at 0x8b7f898>
```



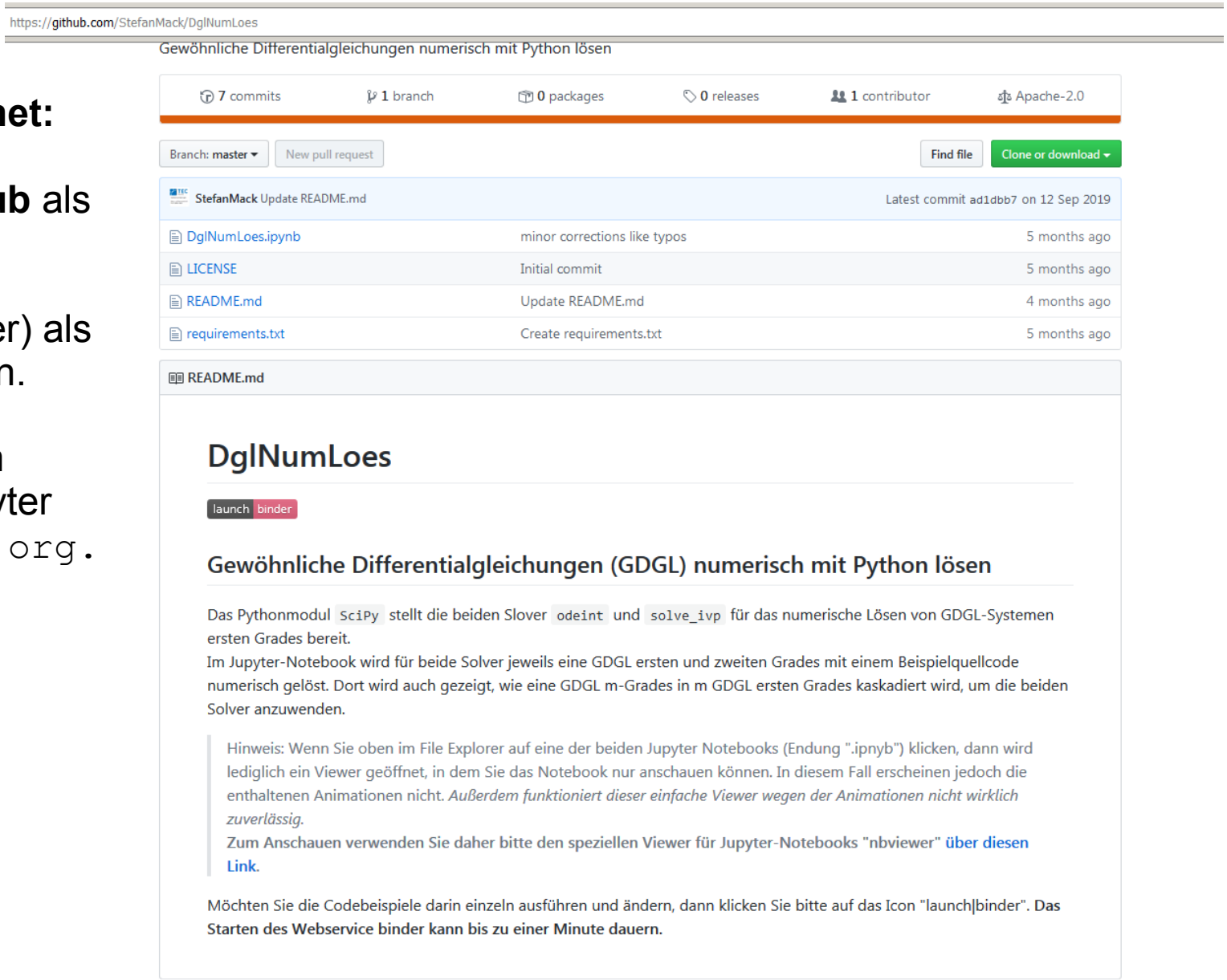
Beispiel Jupyter Notebook auf GitHub

Für Bereitstellung im Internet:

- Verwendung von z.B. **GitHub** als Repository.

- **nbwiewer** (Notebook Viewer) als Webservice zum Visualisieren.

- **Binder** als Webservice zum interaktiven Arbeiten mit Jupyter Notebook. Siehe `mybinder.org`.

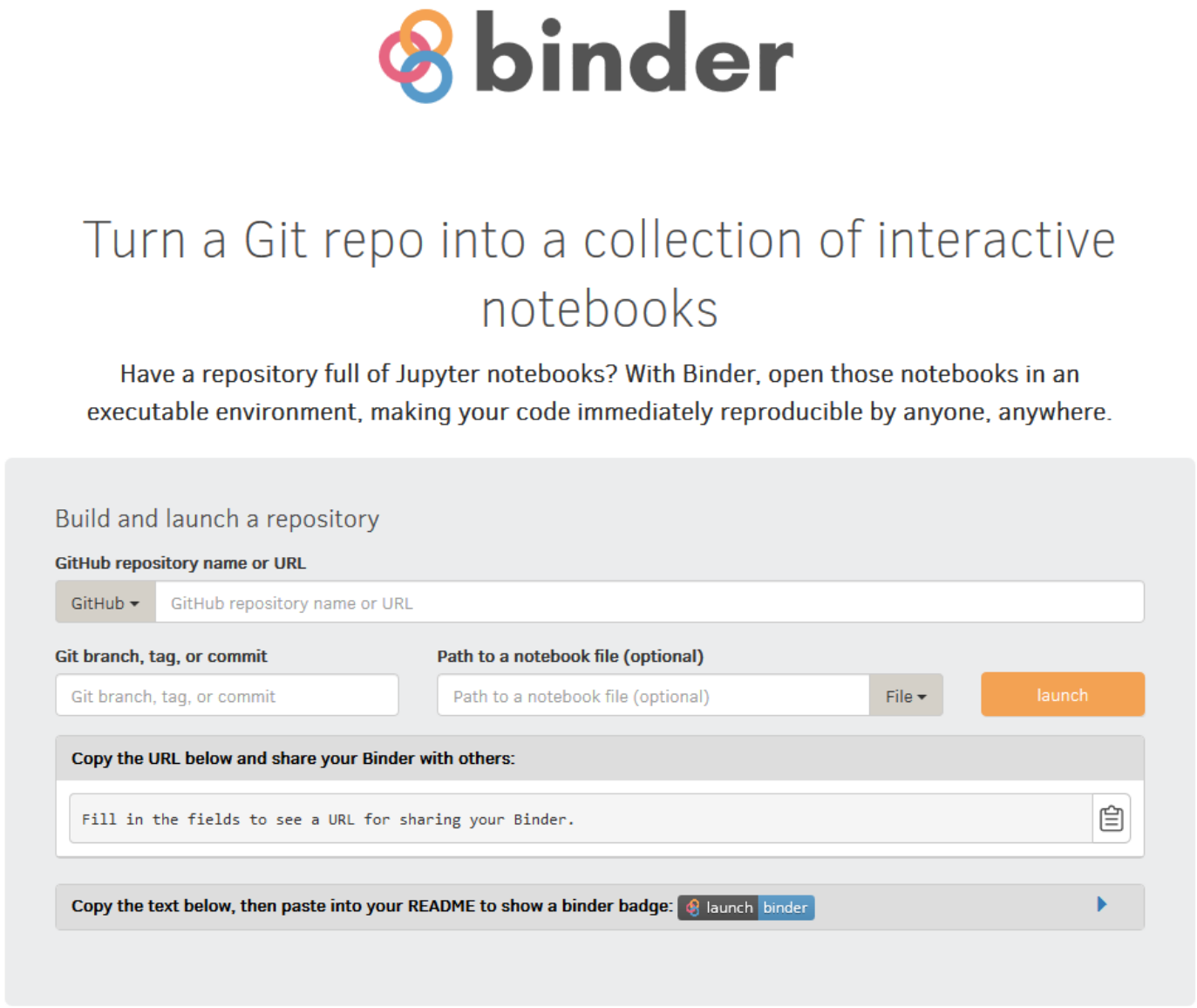


Jupyter Notebook mit Binder öffnen

Startet etwas langsam. Dann aber nahezu gleiche Funktionalität wie Jupyter Notebook lokal ausgeführt auf eigenem PC.

» Lernende können ohne eigene Installation Lehrmaterial im Notebook durcharbeiten, Code entwickeln und testen ...

Webservices **nbviewer** und **Binder** sind Open Source: Damit auch auf (Hoch)Schulserver einsetzbar.



Scientific Python: Matrixalgebra

Modul (=Bibliothek) `NumPy` für Rechnen mit Vektoren und Matrizen

Matrizen als `ndarray`-Objekt, mit unterschiedlichen Datentypen wie `int64`, `float32`, `Bool`, `complex256`.

(Python selbst ohne `NumPy` beherrscht schon komplexe Zahlen!)

Matrixobjekte besitzen Methoden und Attribute. Matrizenrechnung mit Klassenmethoden.
Slicing wie bei MATLAB, jedoch Index beginnt bei Null.

```
In [1]: import numpy as np

        matrix = np.array([[1, 2], [3, 4], [5, 6]])
        type(matrix)

Out[1]: numpy.ndarray

In [2]: matrix * 2

Out[2]: array([[ 2,  4],
               [ 6,  8],
               [10, 12]])

In [3]: np.ones?

In [4]: eins = np.ones((3,2))
        print(eins)

        [[1. 1.]
         [1. 1.]
         [1. 1.]]

In [5]: matrix < 4*eins

Out[5]: array([[ True,  True],
               [ True, False],
               [False, False]])

In [6]: matrix + eins

Out[6]: array([[2., 3.],
               [4., 5.],
               [6., 7.]])
```

```
Signature: np.ones(shape, dtype=None, order='C')
Docstring:
Return a new array of given shape and type, filled with ones.

Parameters
----------
shape : int or sequence of ints
    Shape of the new array, e.g., ``(2, 3)`` or ``2``
```

Scientific Python: Symbolische Mathematik
Python = Computeralgebrasystem

Bibliothek `SymPy` zusammen mit Python IDE = Computeralgebrasystem (CAS)

In [1]:
```
import sympy
sympy.init_printing()
x = sympy.Symbol("x")
y = sympy.Symbol("y")
```

In [2]:
```
expr1 = sympy.sin(x + y)
expr1
```

Out[2]: $\sin(x + y)$

In [3]:
```
sympy.expand(expr1, trig=True)
```

Out[3]: $\sin(x)\cos(y) + \sin(y)\cos(x)$

In [4]:
```
expr2 = 2 * (x**2 - x) - x * (x + 1)
expr2
```

Out[4]: $2x^2 - x(x+1) - 2x$

In [5]:
```
expr2.simplify()
```

Out[5]: $x(x-3)$

In [6]:
```
expr2.diff(x)
```

Out[6]: $2x - 3$

In [7]:
```
expr2.integrate(x)
```

Out[7]: $\frac{x^3}{3} - \frac{3x^2}{2}$

In [8]:
```
n = sympy.symbols("n", integer=True)
```

In [9]:
```
from sympy import oo
expr3 = sympy.Sum(1/(n**2), (n, 1, oo))
expr3
```

Out[9]: $\sum_{n=1}^{\infty} \frac{1}{n^2}$

In [10]:
```
a, b, c = sympy.symbols("a, b, c")
```

In [11]:
```
expr4 = a * x**2 + b * x + c
sympy.solve(expr4, x)
```

Out[11]: $\left[\frac{-b + \sqrt{-4ac + b^2}}{2a}, -\frac{b + \sqrt{-4ac + b^2}}{2a}\right]$

In [12]:
```
d = sympy.symbols("d")
```

In [13]:
```
expr5 = sympy.Matrix([[a, b], [c, d]])
expr5
```

Out[13]: $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$

In [14]:
```
expr5 * expr5
```

Out[14]: $\begin{bmatrix} a^2 + bc & ab + bd \\ ac + cd & bc + d^2 \end{bmatrix}$

Quelle:
nbviewer.jupyter.org/github/jrjohansson/numerical-python-book-code/blob/master/ch03-code-listing.ipynb

Scientific Python: Datenvisualisierung

Bibliothek `Matplotlib`:
Populärste Pythonbibliothek mit starker Anlehnung an MATLAB für 2D und 3D Visualisierung.

Objektorientiert, für MATLAB-Umsteiger auch mit MATLAB-Syntax verwendbar.

Optimal für Plots in Präsentationen oder Abschlussarbeiten.

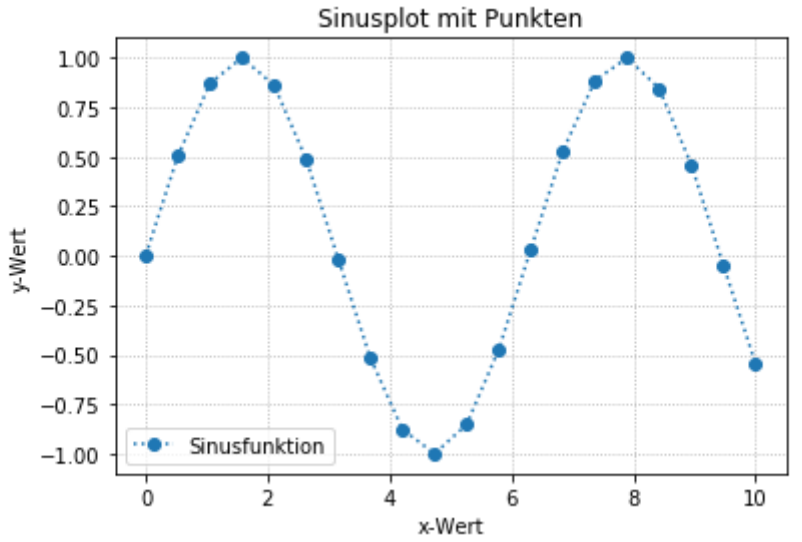
Alternative Python-Bibliotheken:
`Seaborn` (Statistik) ()
`Plotly` (Web) (`plot.ly/python/`)

Rendern innerhalb Jupyter Notebook über `%matplotlib inline`.

Plots auch in separatem Fenster mit Zoomfunktion möglich (Pixelzugriff bei Bildverarbeitung).

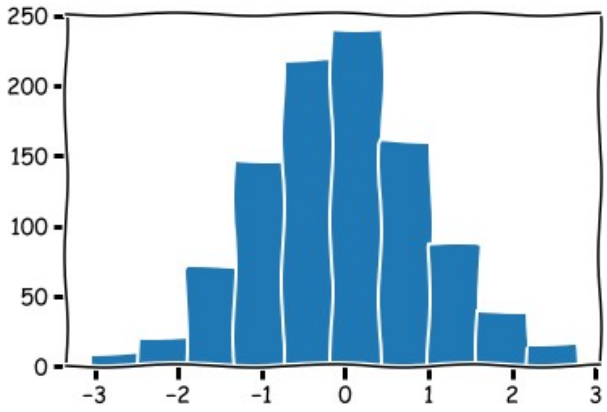
```
In [1]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 20)
y = np.sin(x)
# Darstellung (fig) und Plot (ax) instanzieren
fig, ax = plt.subplots()
# gestrichelte Linie und runde Symbolpunkte
ax.plot(x,y,':o',color='C0', label='Sinusfunktion');
ax.set_xlabel('x-Wert') # Beschriftung x-Achse
ax.set_ylabel('y-Wert')
ax.set_title('Sinusplot mit Punkten')
ax.grid(linestyle=':') # Gitter erzeugen
ax.legend() # Legende mit Werten aus ax.plot(label='')
```

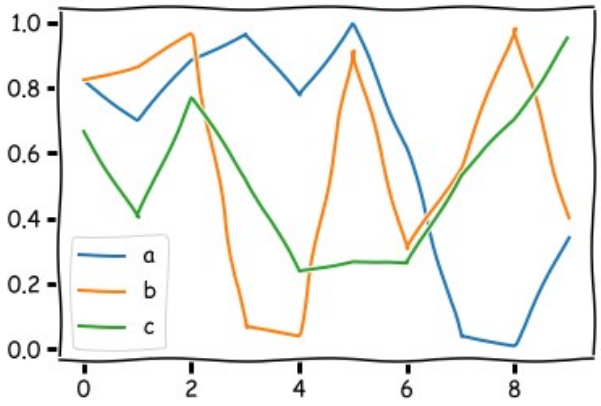
```
Out[1]: <matplotlib.legend.Legend at 0x857bd08>
```

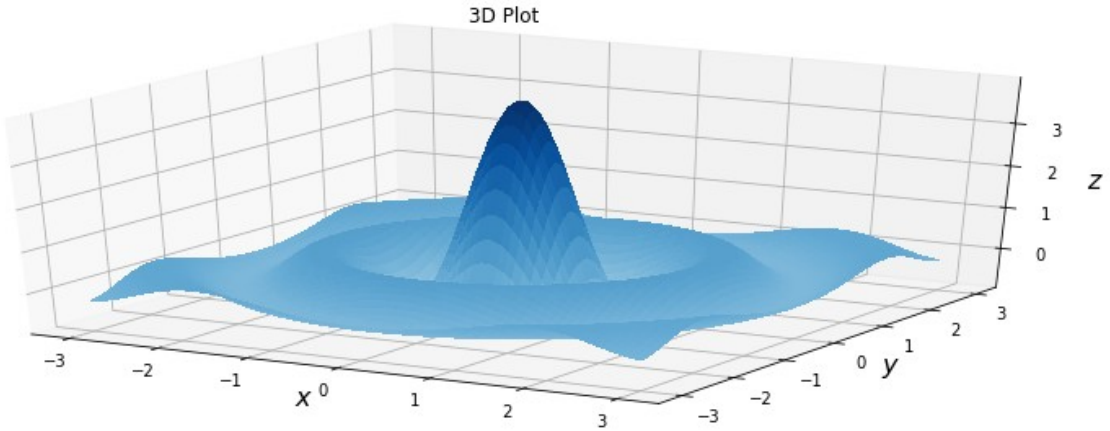


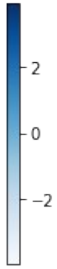
Scientific Python: Datenvisualisierung

Sehr viele verfügbare „Styles“ wie dieser Comic-artige









Viele verfügbare Colormaps für 3D und Surface Plots

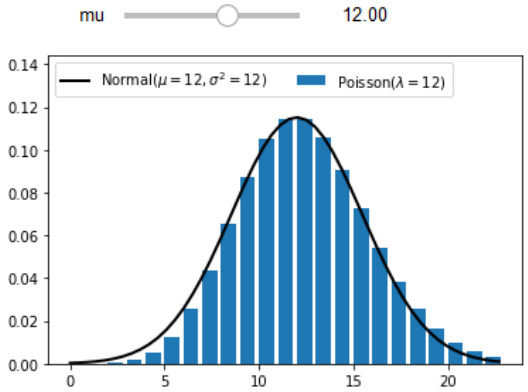
Quellen:

`nbviewer.jupyter.org/github/jrjohansson/numerical-python-book-code/blob/master/ch03-code-listing.ipynb`

`nbviewer.jupyter.org/github/StefanMack/Matplotlib/blob/master/matplotlibOop.ipynb`

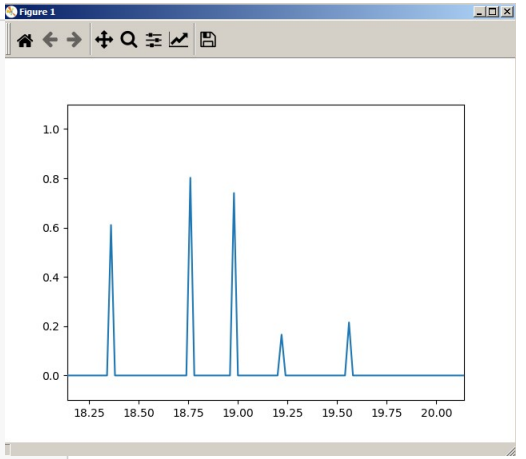
Scientific Python: Animierte Grafik

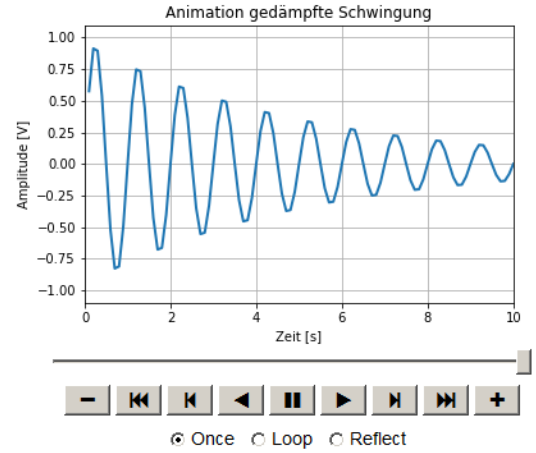
```
In [1]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

def f(mu):
    X = stats.norm(loc=mu, scale=np.sqrt(mu))
    N = stats.poisson(mu)
    x = np.linspace(0, X.ppf(0.999))
    n = np.arange(0, x[-1])

    fig, ax = plt.subplots()
    ax.plot(x, X.pdf(x), color='black', lw=2, label="Normal($\mu=%d, \sigma^2=%d$)" % (mu, mu))
    ax.bar(n, N.pmf(n), align='edge', label=r"Poisson($\lambda=%d$)" % mu)
    ax.set_ylim(0, X.pdf(x).max() * 1.25)
    ax.legend(loc=2, ncol=2)
    plt.close(fig)
    return fig

from ipywidgets import interact
import ipywidgets as widgets
interact(f, mu=widgets.FloatSlider(min=1.0, max=20.0, step=1.0));
```



Vergleich Gauß-/Poisson-Verteilung. Jupyter Notebook mit `ipywidgets`

Quelle: `nbviewer.jupyter.org/github/jrjohansson/numerical-python-book-code/blob/master/ch01-code-listing.ipynb`





Animation Oszi, gedämpfte Schwingung. Jupyter Notebook oder IDE mit `matplotlib.animation`.

Quelle: `nbviewer.jupyter.org/github/StefanMack/Matplotlib/blob/master/matplotlibAnimation.ipynb`

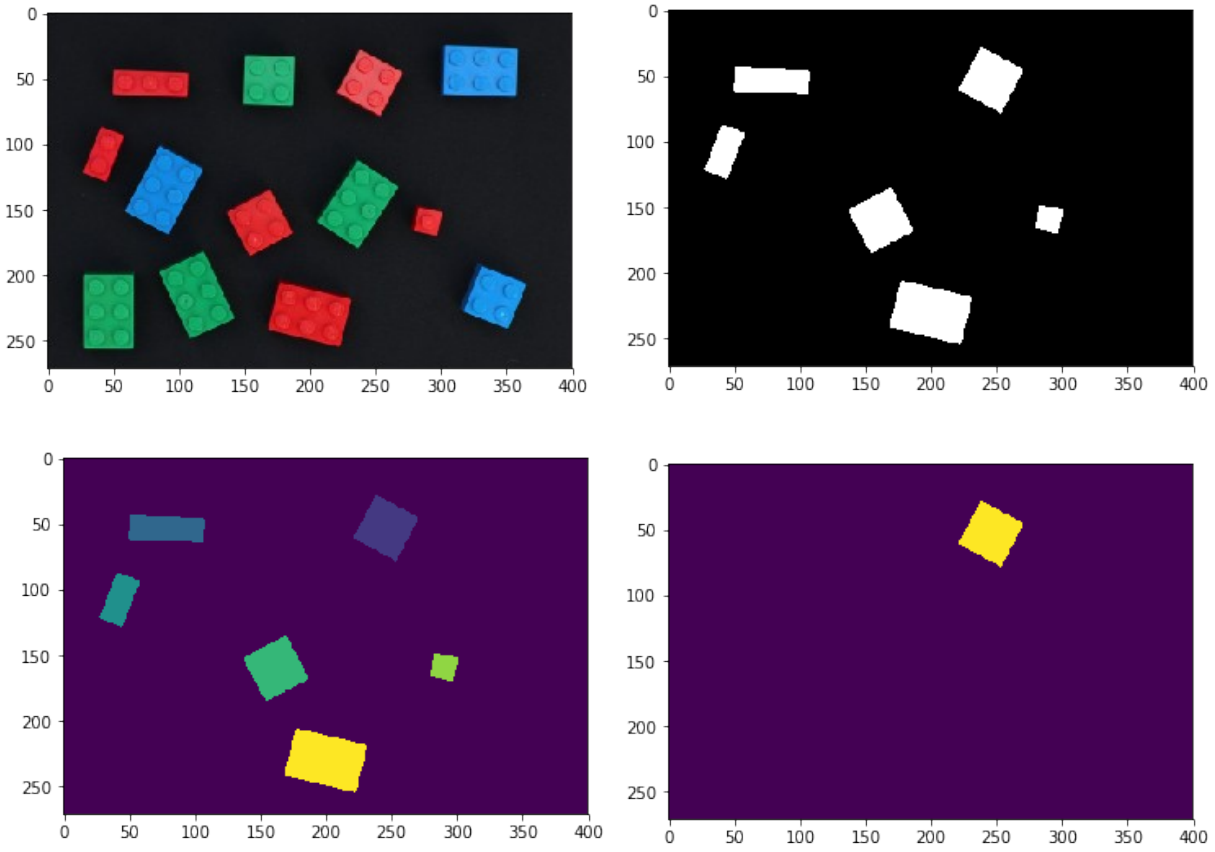
Scientific Python: Bildverarbeitung

`PythonCV` als „Wrapper“ der C-Bibliothek `OpenCV`.

Einfaches Beispiel:

Rote Legosteine herausfiltern, diese zählen und deren Positionen bestimmen.
(Praktikum Hochschule Reutlingen, Bachelor Mechatronik)

- Pixelklassifizierung
- Labeling
- Merkmalsextraktion



```
In [18]: BrickOne = (labels == 1).astype(dtype='uint8') # uint8-Matrix erstellen
         plt.imshow(BrickOne)
         BrickOneMom = cv2.moments(BrickOne) # Berechnung der Momente
         cX = int(BrickOneMom["m10"] / BrickOneMom["m00"])
         cY = int(BrickOneMom["m01"] / BrickOneMom["m00"])
         print('Legostein Nr. 1: x = {0}, y = {1}'.format(cX,cY))

         Legostein Nr. 1: x = 246, y = 53
```

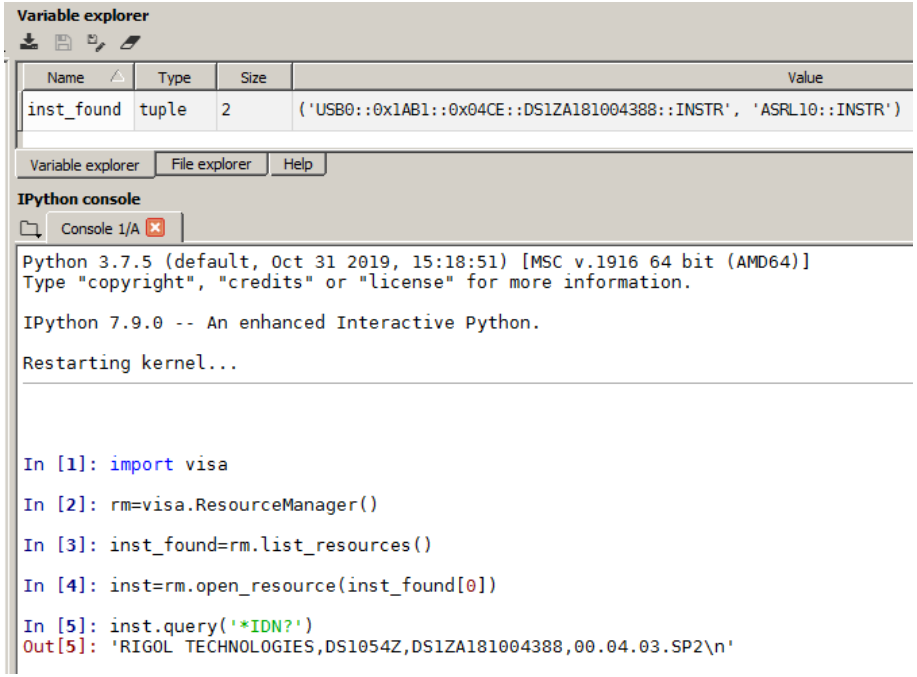
Quelle:

`nbviewer.jupyter.org/github/StefanMack/PraktMesstBV/blob/master/2_OpenCVPython-ImageProcessing.ipynb`

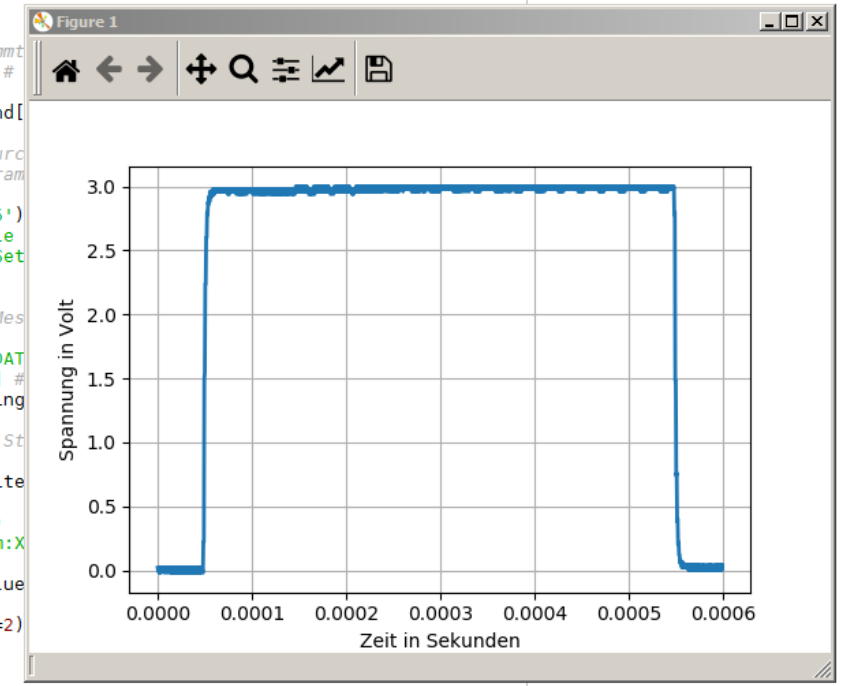
Scientific Python: Messautomatisierung

Studierende erhalten Jupyter Notebook mit Code für automatisierte Messung mit Oszilloskop Marke A.

Aufgabe im Praktikum:
In IDE Spyder interaktiv, explorativ mit Oszilloskop Marke B einen Code mit gleicher Funktionalität entwickeln.




```
import visa
import matplotlib.pyplot as plt
import numpy as npy
from time import sleep

rm=visa.ResourceManager() # Kommt
inst_found=rm.list_resources() #

inst=rm.open_resource(inst_found[

inst.write(':AUT') # Autoset durc
sleep(5) # Nötig da sonst Program

inst.write(':CHANnel1:SCALe 0.5')
inst.write(':TIMebase:MAIN:SCALe
inst.write(':TIMebase:MAIN:OFFSet


inst.write(':WAV:FORM ASC') # Mes

values_raw = inst.query(':WAV:DAT
values_string = values_raw[11:] #
values_string_sep = values_string

values_float = [] # Umwandlung St
for item in values_string_sep:
    values_float.append(float(ite

y_val = npy.array(values_float)
delta_t = inst.query(':WAVeform:X
dt=float(delta_t)
x_val = npy.arange(0.0, len(value

plt.plot(x_val,y_val,linewidth=2)
plt.xlabel('Zeit in Sekunden')
plt.ylabel('Spannung in Volt')
plt.grid(True)
plt.savefig('my_plot.png')
plt.show()

inst.close()
```



Mit Bibliothek `PyVisa` fast alle moderneren Messgeräte steuerbar und auslesbar.

Quelle: `github.com/StefanMack/PraktMessVISA`

Scientific Python: Audiosignalerzeugung

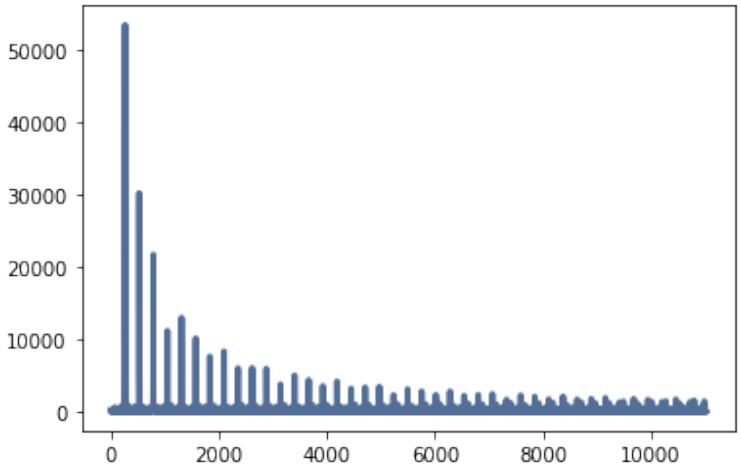
Nutzung der Soundkarte des PC für Audiosignale

Hier: Simulation eines Schulorchesters
(..als Motivation für digitale Signalverabeitung)

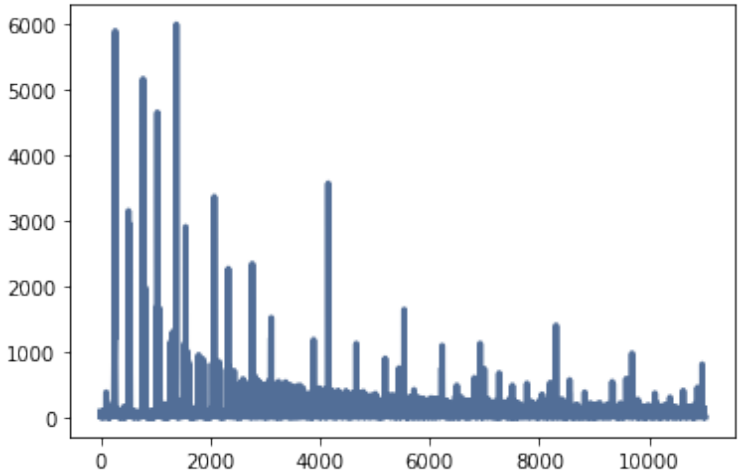
... funktioniert auch mit Web Service Binder!

```python
def random_freq(midi_num):

    # simulate poor tuning by adding gaussian noise to the MIDI number
    midi_num += random.gauss(0, 0.5)

    # one kid out of 10 plays the wrong note
    if random.random() < 0.1:
        midi_num += random.randint(-5, 5)

    freq = midi_to_freq(midi_num)

    # and one kid in 10 pops an overtone
    if random.random() < 0.1:
        freq *= random.randint(2, 5)

    return freq
```

Spektrum Ton C:
Gutes Schulorchester



Schlechtes Schulorchester



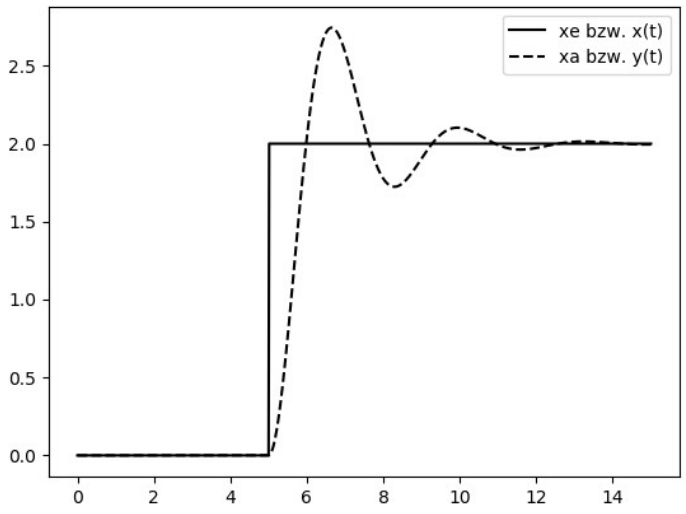
Quelle: `allendowney.github.io/ThinkDSP/tutorial.html`

Scientific Python: DGLs numerisch lösen

Pythonbibliothek `SciPy`: Beispiel Einschwingvorgang PT2-System
(konkret Eingang Messgerät mit parasitärer Induktivität und Kapazität)

Empfindlichkeit E
Dämpfungsgrad D
Zeitkkonstante T
Eingangssignal xe
Ausgangssignal xa

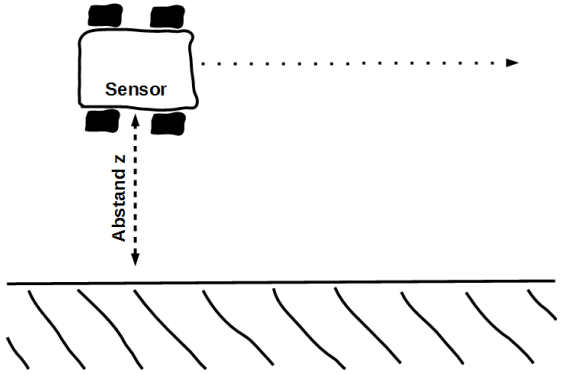
$$T^2 \frac{d^2}{dt^2} x_a(t) + 2DT \frac{d}{dt} x_a(t) + x_a(t) = E\, x_e(t)$$




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

# %% Importierte Module
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp

# %% Unabhängige (Erreger-) Funktion xe(t) und die Funktion mit Ableitungen f(t)
def xe(t):
    x = 0
    if (t>5):
        x = 2
    return x

def f(t, y_array, E,T,D):
    y,dydt = y_array
    dy2dt2 = E/T**2 * xe(t) - 2.0*D/T * dydt - 1/T**2 * y
    return [dydt,dy2dt2]

# %% Zeitpunkte und Anfangswerte festlegen
tspan = np.linspace(0, 15, 1000)
yinit = [0,0] # Anfangswert für xa und d/dt xa

# %% DGL numerisch lösen
sol = solve_ivp(lambda t, y_array: f(t, y_array, E=1.0, T=0.5, D=0.3), [tspan[0], tspan[-1]], yinit, t_eval=tspan)

# %% xe und xa (y) grafisch darstellen
x_vals = np.zeros(np.size(sol.t))
for index,time in enumerate(sol.t):
    x_vals[index] = xe(time)

fig, ax = plt.subplots()
ax.plot(sol.t, x_vals, 'k-', label='xe bzw. x(t)')
ax.plot(sol.t, sol.y[0,:], 'k--', label='xa bzw. y(t)')
ax.legend(loc='best')
```

Quelle: `github.com/StefanMack/DglNumLoes`

Scientific Python: Signalverarbeitung Beispiel Kalman-Filter



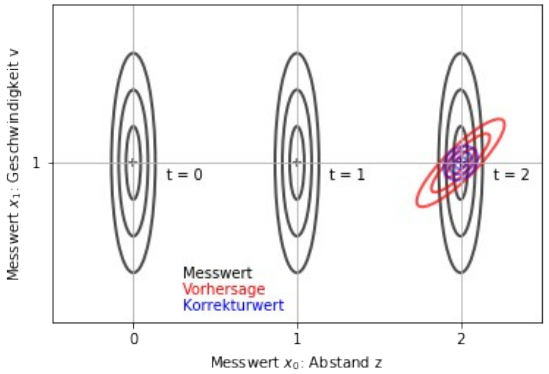
Ziel: Roboter mit nur einem einzelnen Abstandssensor fährt parallel zur Wand.

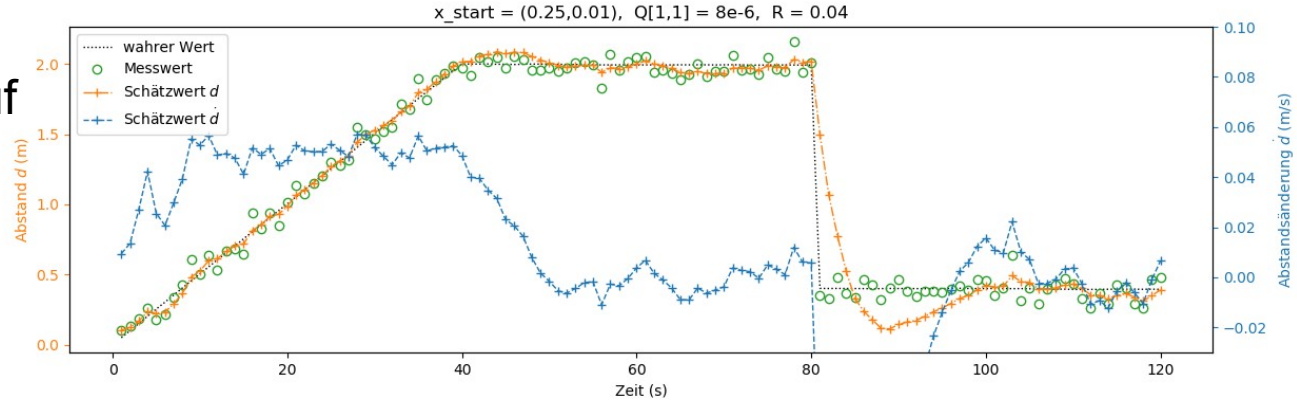
Pythonbibliotheken `NumPy` und `SciPy`

Projekt Master Mechatronik:

Reihe von zehn Jupyter Notebooks:

- Entwicklung Theorie Kalman-Filter.
- Jeder Algorithmus in Python implementiert und an Simulationsdaten getestet.
- Studierende beobachten Einfluss verschiedener Filterparameter.
- Anschließend Implementierung auf BeagleBone-Steuerung in einem Roboterfahrzeug sowie Entwicklung einer Regelung.





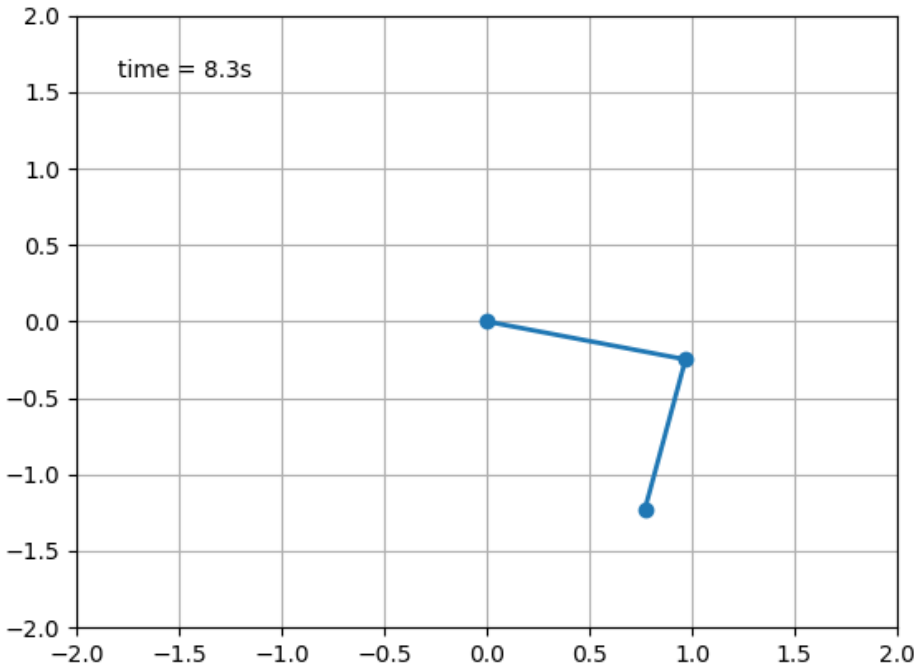
Quelle: `github.com/StefanMack/KalmanSensys`

Scientific Python kann noch viel mehr...

- Numerische Mathematik
  Kombiniert mit Animation
  = Simulation
  (alternativ Animation mit pygame)
- Machine Learning
- Statistik
- Signalverarbeitung

Am besten einfach in
„A gallery of interesting
Jupyter Notebooks“,
auf GitHub oder irgendwo
im Internet stöbern...

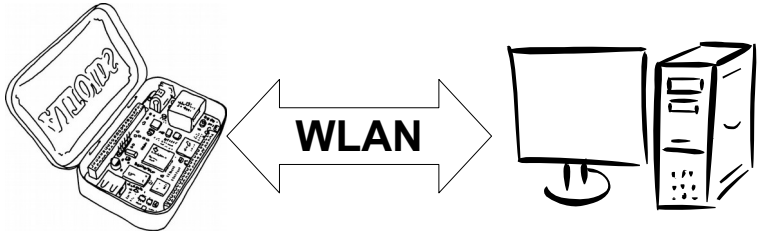
Beispiel Simulation Doppelpendel mit Animation



Quelle: matplotlib.org/examples/animation/double_pendulum_animated.html

Physical Computing mit Python

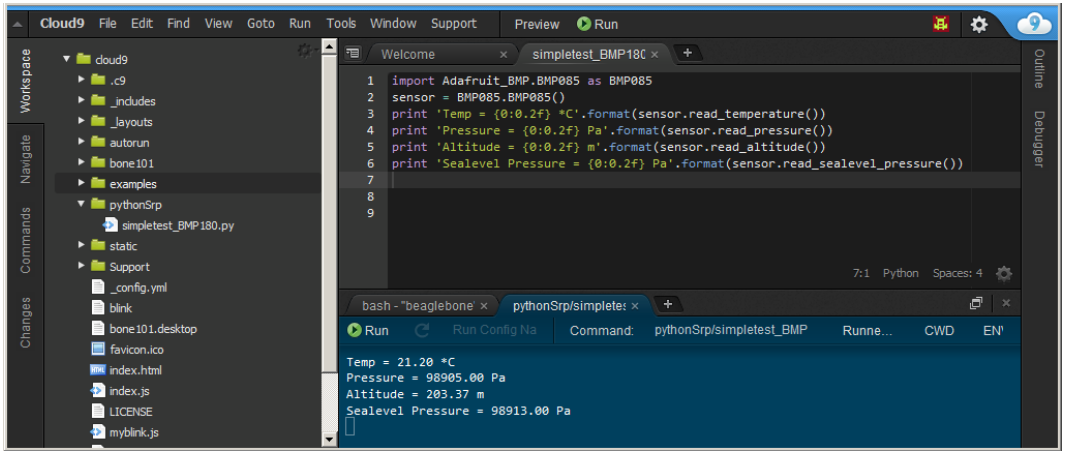
Physical Computing: Software kommuniziert mit Sensoren und Aktoren
Beispiel: „Steuern-Messen-Regeln“:
Auslesen Sensoren, Ansteuern Motoren,
Kommunikation mit anderen Computern,...



Einplatinencomputer

PC, Tablet,
Smartphone

Einplatinencomputer wie Raspberry Pi (RasPi)
oder BeagleBone standardmäßig mit Python.



Cloud9 IDE im PC-Browser

Arbeiten mit Python über

- Bash-Konsole
- z.B. IDE Thonny lokal auf RasPi
- Cloud9 IDE (Webservice) auf RasPi via Netz + PC-Browser (optimal für Robotikprojekte)
- Jupyter Notebook auf RasPi via Netz + PC-Browser

Python auf SOC + FPGA für Messtechnikanwendung

Einplatinencomputer erweitert mit FPGA » performante Messtechnik:

- AD-Wandler mit >100 MSa/a (Oszilloskop)
- Arbiträrfunktionsgenerator bis MHz
- Netzwerkanalysator
- Logikanalysator
- Software Defined Radio
- Datenlogger

... Zugriff auf diese Funktionalitäten jeweils über Python-Skripte oder Jupyter Notebooks (STEMLab) ausgeführt auf Einplatinencomputer.

ADALM2000, ca. 100 €

Bildquelle: analog.com

STEMLab, ca. 200 €

Bildquelle: reichelt.de

Micropython auf einem Mikrocontroller

STEMLab: 230 € ADALM2000: 110 € BeagleBone: 50 € RasPi: 40 €

Physical Computing mit Python geht auch mit dem ESP32: 10 €!

Wieso jetzt ein Mikrocontroller (µC) mit Python-Interpreter? Ohne Betriebssystem?

Python-Interpreter = virtuelle Maschine. µC baucht nur ausreichend Speicher und Taktrate dafür

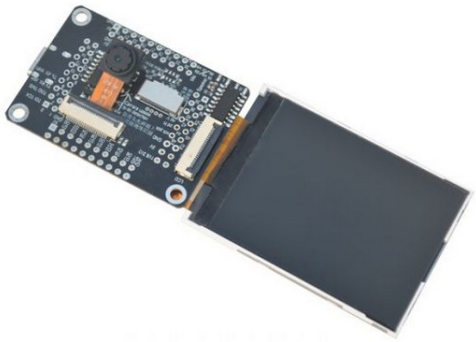
» „Flashen“ des µC mit Python-Interpreter = **Micropython**

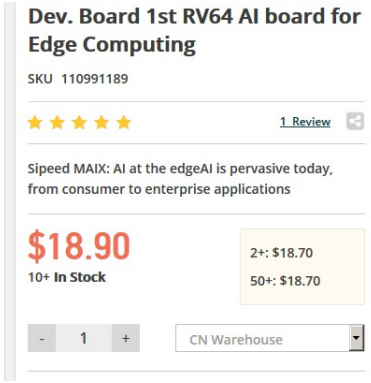
Funktionalitäten ESP32 µC-Platine unter Micropython:

WiFi, Timer, GPIOs, ADC, PWM, SPI, I2C, RTC, Deep-Sleep, Board interne Sensoren,...

Außer ESP32 noch viele weitere µC-Platinen Micropython-fähig, teils mit extra KI-/Signalverabeitungsprozessoren.

Bemerkenswert:
Chinesische Eigenentwicklung
Sipeed MAIX (Dual Core RISC V, 400 MHz, 8 MB Ram)



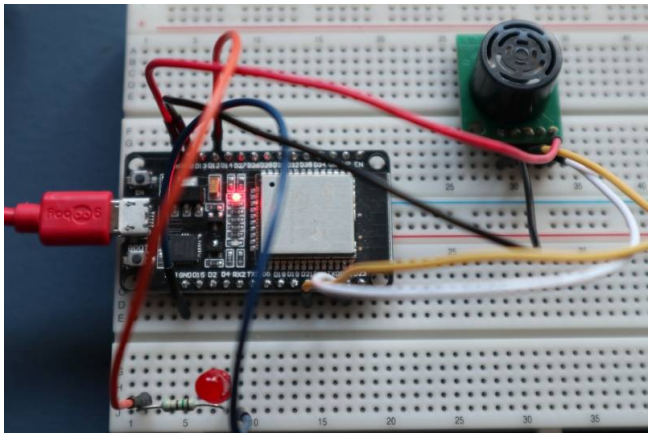


Bildquelle: seeedstudio.com

Micropython mit ESP32

IDE Thonny (u.a.) unterstützt Micropython Interpreter und Dateitransfer zwischen PC und µC.

Beispiel:
I²C- Utraschallsensor SRF02 auslesen und messwertabhängig LED schalten.



Thonny - MicroPython device :: /SRF02ReadBlink.py @ 20 : 59

File Edit View Run Device Tools Help

Files

MicroPython device

- blink.py
- boot.py
- SRF02Read.py
- SRF02ReadBlink.py
- touchSensor.py

This computer
C: \ Daten \ work \ uPython \ workSpace \ esp32

- analogRead.py
- analogReadAttenuation.py
- blinkPin15.py
- boot.py
- DemoMesstechnikVorlesung
- esp32WlanTest.py
- leer.py
- main.py
- marioMusik.py
- pwmBeispiel.py
- RealTimeClock.py
- SRF02Read.py
- ssd1306.py
- test_0.py
- test_1.py
- test_2.py

<untitled> [SRF02ReadBlink.py]

```
1  import machine
2  import time
3
4  pin12 = machine.Pin(12, machine.Pin.OUT)
5  i2c = machine.I2C(scl=machine.Pin(21), sda=machine.Pin(22))
6  for adress in i2c.scan():
7      print('Sensor gefunden an Adresse {}'.format(hex(adress)))
8
9  for i in range(10):
10     i2c.writeto_mem(0x70, 0x00, b'\x51')
11     time.sleep(1)
12     range_raw=i2c.readfrom_mem(0x70, 0x02, 2)
13     range_cm = range_raw[0]*256 + range_raw[1]
14     if range_cm < 30:
15         pin12.value(1)
16     else:
17         pin12.value(0)
18
19     print('Rohmessdaten: 0x{:02x} 0x{:02x}'.format(range_raw[0],range_raw[1]))
20     print('Gemessener Abstand in cm: {}'.format(range_cm))
```

Shell

```
I (0) cpu_start: Starting scheduler on APP CPU.
MicroPython v1.11-422-g98c2eabaf on 2019-10-11; ESP32 module with ESP32
Type "help()" for more information.

MicroPython v1.11-422-g98c2eabaf on 2019-10-11; ESP32 module with ESP32
Type "help()" for more information.
>>> help()

  Welcome to MicroPython on the ESP32!

  For generic online docs please visit http://docs.micropython.org/
```

Selbst WLAN-Kommunikation mit ESP32-Micropython kinderleicht!

Fehlt noch was?

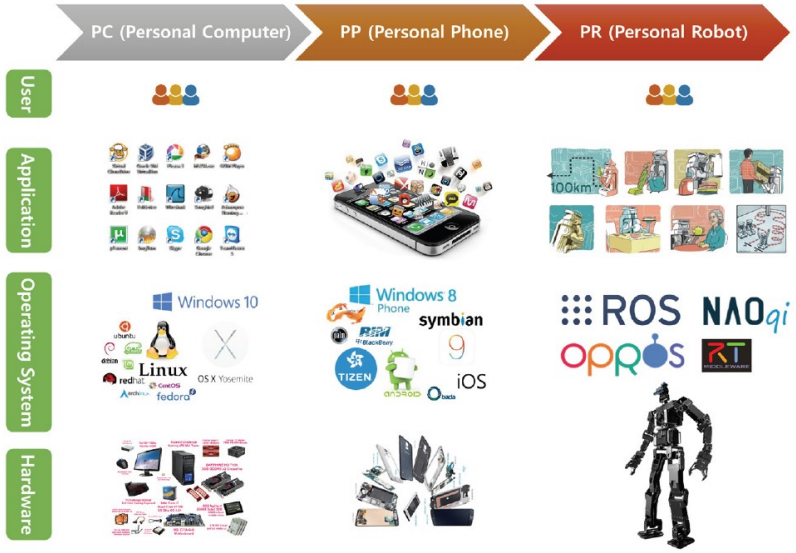
Komplexe Robotersysteme über **ROS** (Robot Operation System) steuern:
Knoten (=Softwaremodule für die einzelnen Komponenten) sind alternativ in C++ oder Python programmierbar.

`PyGame`: Bibliothek zur Erstellung von Spielen mit 2D bzw. 3D Grafik. Ideal für Kinematiksimulation.
Siehe z.B.:
`github.com/willelson/Pygame-physics`
`github.com/tomlinsonk/pygame-physics`

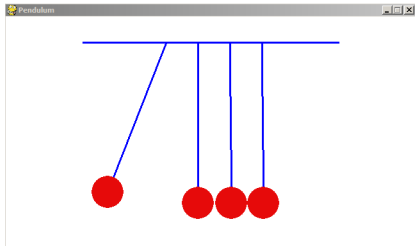
`Cython`: Zum Beschleunigen von Python-Skripten via Übersetzen des Python-Codes in C/C++ und Kompilieren zu Laufzeitbibliothek.
Siehe:
`http://docs.cython.org/en/latest/src/quickstart/overview.html`

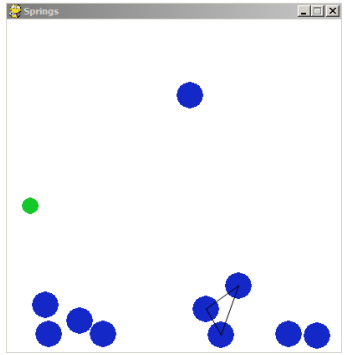
Python auf **Lego Mindstorms**

Bildquelle: Y.S. Oyo et al.: ROS Robot Programming.



Robotik wird sich ähnlich entwickeln wie PC- und Smartphonetechnik.





PyGame: Physikalische Simulation hüpfender Bälle, Kugelstoßpendel

Auswahl an Literatur und Internetlinks

Internetquellen:
Distribution Anaconda: `anaconda.com/distribution/`
Distribution WinPython: `winpython.github.io/`
Python Einführungskurs Uni Heidelberg: `physi.uni-heidelberg.de/Einrichtungen/AP/Python.php`
Python Code Style Guide: `python.org/dev/peps/pep-0008/`
Datenvisualisierung mit matplotlib: `matplotlib.org`
Symbolisches Rechnen mit SymPy: `sympy.org/en/index.html`
Audiosignalverarbeitung: `allendowney.github.io/ThinkDSP/tutorial.html`
OpenCV 3 Python Tutorials: `docs.opencv.org/3.1.0/d6/d00/tutorial_py_root.html`
MicroPython mit ESP32: `docs.micropython.org/en/latest/esp32/tutorial/intro.html`

Jupyter Notebooks:
Projektseite Jupyter: `jupyter.org`
J. VanderPlas: Python Data Science Handbook. `github.com/jakevdp/PythonDataScienceHandbook`
Jupyter Notebook Gallery: `github.com/jupyter/jupyter/wiki/A-gallery-of-interesting-Jupyter-Notebooks`
Markdown Sprache: `daringfireball.net/projects/markdown/`
Lernmaterialien S. Mack Hochschule Reutlingen: `github.com/StefanMack?tab=repositories`

Fachbücher:
H.-W. Philippsen: Einstieg in die Regelungstechnik mit Python. Hanser-Verlag. Webseite: `einstieg-rt.de`
C. Rossant: IPython Cookbook. Packt Verlag. Webseite: `github.com/ipython-books/cookbook-2nd`
R. Johansson: Numerical Python. Apress-Verlag. Webseite: `jrjohansson.github.io/numericalpython.html`
T. Häberlein: Informatik. Eine praktische Einführung mit Bash und Python. De Gruyter-Verlag.
H.-B. Woyand: Python. Für Ingenieure und Naturwissenschaftler. Hanser-Verlag.
S. Kamminski: Python 3. De Gruyter-Verlag.
B. Klein: Einführung in Python 3. Hanser-Verlag.
J. Ernesti, P. Kaiser: Python 3. Das umfassende Handbuch. Rheinwerk-Verlag.
J. Unipogo: Python for Signal Processing. Springer-Verlag.