

- ▶ Wer aktiv mitmachen möchte, setzt sich bitte nach vorn.
- ▶ Einen USB-Stick mit Anaconda hab ich dabei
- ▶ Alle Folien & Beispiele auch auf dem Stick
- ▶ Handies bitte auf lautlos stellen
- ▶ oder unter `https://github.com/UweZiegenhagen/PythonAndLaTeX`

Python & L^AT_EX

Dante e. V. Frühjahrstagung 2017

Dr. Uwe Ziegenhagen

22. März 2017

Überblick

Was machen wir heute?

- ▶ Python Grundlagen
- ▶ Python in \LaTeX Dokumenten
- ▶ Erzeugung von \LaTeX Dokumenten

Voraussetzungen

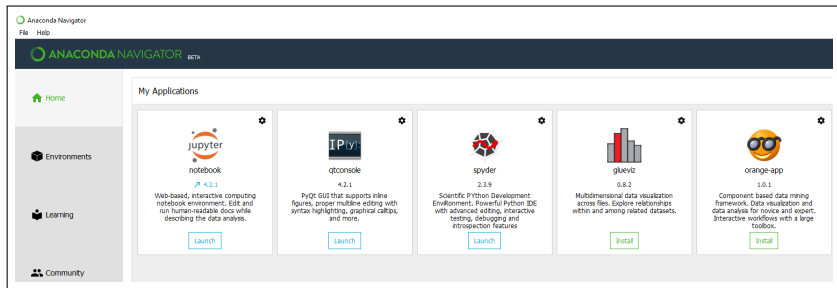
Was wird benötigt

- ▶ Aktuelle T_EX Installation (T_EX Live 2016)
- ▶ Python3 Installation, vorzugsweise Anaconda
- ▶ Pakete:
 - ▶ numpy
 - ▶ jinja2
 - ▶ pandas
- ▶ Diese Folien und Code-Beispiele unter:

<https://github.com/UweZiegenhagen/PythonAndLaTeX>

„Scientific Python“ Distributionen

- ▶ Linux/MacOS X kommen mit Python, aber nicht SciPy
- ▶ Manuell nachinstallieren oder „echte“ Installation
- ▶ Meine Empfehlung: Anaconda
 - ▶ Anaconda (<https://www.continuum.io/downloads>)
 - ▶ WinPython (<https://winpython.github.io>)



Das SciPy Framework

SciPy-Distributionen enthalten:

NumPy Matrizen, Vektoren, Algorithmen

IPython Matlab/Mathematica-ähnliche Umgebung

Matplotlib Wiss. Grafik, Basis für die seaborn Bibliothek

SymPy Symbolische Mathematik

... etc, etc.

Testen der Installation

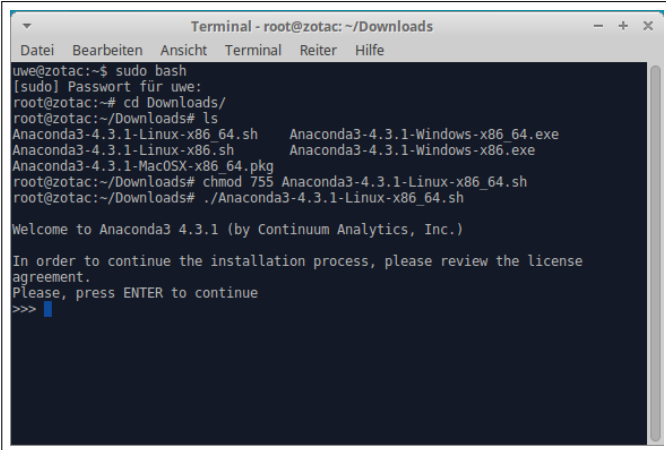
- Funktionieren die folgenden Befehle?

```
1 import jinja2
2 import pandas
3 import numpy
```

Wenn ja, dann gut, wenn nein, dann

- entweder Anaconda installieren :-)
- oder mittels pip nachinstallieren :-)

Anaconda Installation 1

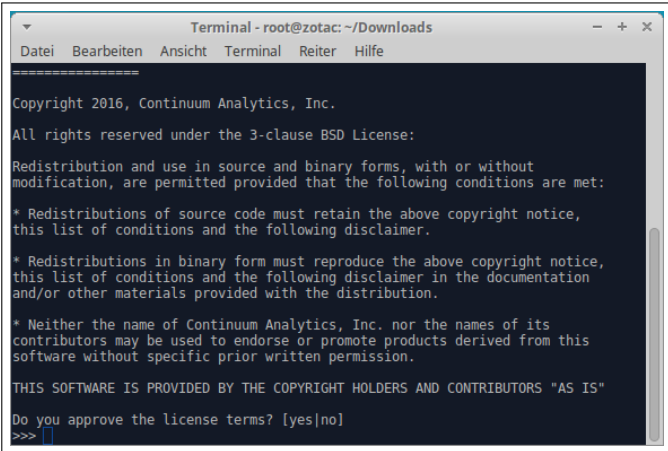
A terminal window titled "Terminal - root@zotac: ~/Downloads" with a menu bar containing "Datei", "Bearbeiten", "Ansicht", "Terminal", "Reiter", and "Hilfe". The terminal shows the following commands and output:

```
uwe@zotac:~$ sudo bash
[sudo] Passwort für uwe:
root@zotac:~# cd Downloads/
root@zotac:~/Downloads# ls
Anaconda3-4.3.1-Linux-x86_64.sh      Anaconda3-4.3.1-Windows-x86_64.exe
Anaconda3-4.3.1-Linux-x86_64.sh      Anaconda3-4.3.1-Windows-x86_64.exe
Anaconda3-4.3.1-MacOSX-x86_64.pkg
root@zotac:~/Downloads# chmod 755 Anaconda3-4.3.1-Linux-x86_64.sh
root@zotac:~/Downloads# ./Anaconda3-4.3.1-Linux-x86_64.sh

Welcome to Anaconda3 4.3.1 (by Continuum Analytics, Inc.)

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>> █
```


Anaconda Installation 2

A terminal window titled "Terminal - root@zotac: ~/Downloads" with a menu bar containing "Datei", "Bearbeiten", "Ansicht", "Terminal", "Reiter", and "Hilfe". The terminal displays the Anaconda license terms, including copyright information for 2016, a 3-clause BSD license, and redistribution rules. It ends with the prompt "Do you approve the license terms? [yes|no]" and a cursor on the next line.

```
=====
Copyright 2016, Continuum Analytics, Inc.

All rights reserved under the 3-clause BSD License:

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.

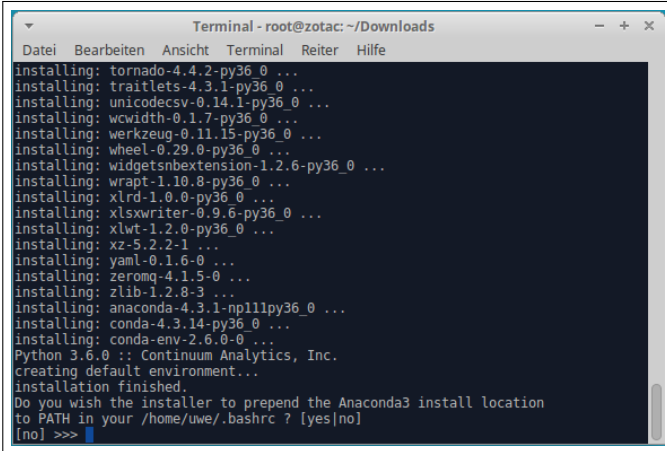
* Redistributions in binary form must reproduce the above copyright notice,
this list of conditions and the following disclaimer in the documentation
and/or other materials provided with the distribution.

* Neither the name of Continuum Analytics, Inc. nor the names of its
contributors may be used to endorse or promote products derived from this
software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"

Do you approve the license terms? [yes|no]
>>> ☐
```

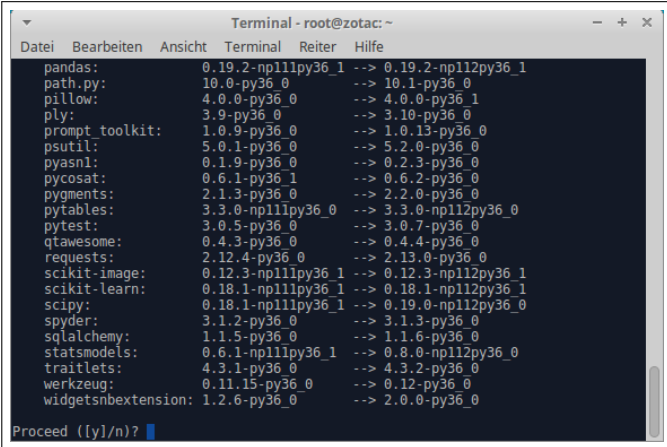
Anaconda Installation 3

A terminal window titled "Terminal - root@zotac: ~/Downloads" with a menu bar containing "Datei", "Bearbeiten", "Ansicht", "Terminal", "Reiter", and "Hilfe". The terminal output shows the installation of various dependencies for Anaconda, including tornado, traitlets, unicodcsv, wcwidth, werkzeug, wheel, widgetsnbextension, wrapt, xlrd, xlswriter, xlwt, xz, yaml, zeromq, and zlib. It then shows the installation of anaconda, conda, and conda-env. The prompt "Python 3.6.0 :: Continuum Analytics, Inc." is displayed, followed by "creating default environment...". The installation is finished, and a question is asked: "Do you wish the installer to prepend the Anaconda3 install location to PATH in your /home/uwe/.bashrc ? [yes|no]". The user has responded with "[no] >>>".

```
Terminal - root@zotac: ~/Downloads
Datei Bearbeiten Ansicht Terminal Reiter Hilfe
installing: tornado-4.4.2-py36_0 ...
installing: traitlets-4.3.1-py36_0 ...
installing: unicodcsv-0.14.1-py36_0 ...
installing: wcwidth-0.1.7-py36_0 ...
installing: werkzeug-0.11.15-py36_0 ...
installing: wheel-0.29.0-py36_0 ...
installing: widgetsnbextension-1.2.6-py36_0 ...
installing: wrapt-1.10.8-py36_0 ...
installing: xlrd-1.0.0-py36_0 ...
installing: xlswriter-0.9.6-py36_0 ...
installing: xlwt-1.2.0-py36_0 ...
installing: xz-5.2.2-1 ...
installing: yaml-0.1.6-0 ...
installing: zeromq-4.1.5-0 ...
installing: zlib-1.2.8-3 ...
installing: anaconda-4.3.1-np111py36_0 ...
installing: conda-4.3.14-py36_0 ...
installing: conda-env-2.6.0-0 ...
Python 3.6.0 :: Continuum Analytics, Inc.
creating default environment...
installation finished.
Do you wish the installer to prepend the Anaconda3 install location
to PATH in your /home/uwe/.bashrc ? [yes|no]
[no] >>>
```

Anaconda Installation 4

`sudo conda update --all`



A terminal window titled "Terminal - root@zotac: ~" displays the output of the command `conda update --all`. The window has a menu bar with "Datei", "Bearbeiten", "Ansicht", "Terminal", "Reiter", and "Hilfe". The output lists various packages and their versions, showing the current version and the version it will be updated to. At the bottom, it asks "Proceed ([y]/n)?".

```
Terminal - root@zotac: ~
Datei Bearbeiten Ansicht Terminal Reiter Hilfe
pandas: 0.19.2-np111py36_1 --> 0.19.2-np112py36_1
path.py: 10.0-py36_0 --> 10.1-py36_0
pillow: 4.0.0-py36_0 --> 4.0.0-py36_1
ply: 3.9-py36_0 --> 3.10-py36_0
prompt_toolkit: 1.0.9-py36_0 --> 1.0.13-py36_0
psutil: 5.0.1-py36_0 --> 5.2.0-py36_0
pyasn1: 0.1.9-py36_0 --> 0.2.3-py36_0
pycosat: 0.6.1-py36_1 --> 0.6.2-py36_0
pygments: 2.1.3-py36_0 --> 2.2.0-py36_0
pytables: 3.3.0-np111py36_0 --> 3.3.0-np112py36_0
pytest: 3.0.5-py36_0 --> 3.0.7-py36_0
qtawesome: 0.4.3-py36_0 --> 0.4.4-py36_0
requests: 2.12.4-py36_0 --> 2.13.0-py36_0
scikit-image: 0.12.3-np111py36_1 --> 0.12.3-np112py36_1
scikit-learn: 0.18.1-np111py36_1 --> 0.18.1-np112py36_1
scipy: 0.18.1-np111py36_1 --> 0.19.0-np112py36_0
spyder: 3.1.2-py36_0 --> 3.1.3-py36_0
sqlalchemy: 1.1.5-py36_0 --> 1.1.6-py36_0
statsmodels: 0.6.1-np111py36_1 --> 0.8.0-np112py36_0
traitlets: 4.3.1-py36_0 --> 4.3.2-py36_0
werkzeug: 0.11.15-py36_0 --> 0.12-py36_0
widgetsnbextension: 1.2.6-py36_0 --> 2.0.0-py36_0
Proceed ([y]/n)?
```

Anaconda Installation 5

Läuft...

```
Terminal - root@zotac: ~
Datei Bearbeiten Ansicht Terminal Reiter Hilfe

pygments: 2.1.3-py36_0 --> 2.2.0-py36_0
pytables: 3.3.0-np111py36_0 --> 3.3.0-np112py36_0
pytest: 3.0.5-py36_0 --> 3.0.7-py36_0
qtawesome: 0.4.3-py36_0 --> 0.4.4-py36_0
requests: 2.12.4-py36_0 --> 2.13.0-py36_0
scikit-image: 0.12.3-np111py36_1 --> 0.12.3-np112py36_1
scikit-learn: 0.18.1-np111py36_1 --> 0.18.1-np112py36_1
scipy: 0.18.1-np111py36_1 --> 0.19.0-np112py36_0
spyder: 3.1.2-py36_0 --> 3.1.3-py36_0
sqlalchemy: 1.1.5-py36_0 --> 1.1.6-py36_0
statsmodels: 0.6.1-np111py36_1 --> 0.8.0-np112py36_0
traitlets: 4.3.1-py36_0 --> 4.3.2-py36_0
werkzeug: 0.11.15-py36_0 --> 0.12-py36_0
widgetsnbextension: 1.2.6-py36_0 --> 2.0.0-py36_0

Proceed ([y]/n)? y

libgcc-5.2.0-0 100% |#####| Time: 0:00:04 244.03 kB/s
alabaster-0.7. 100% |#####| Time: 0:00:00 266.60 kB/s
anaconda-custo 100% |#####| Time: 0:00:00 481.21 kB/s
boto-2.46.1-py 100% |#####| Time: 0:00:06 245.20 kB/s
entrypoints-0. 100% |#####| Time: 0:00:00 278.46 kB/s
greenlet-0.4.1 100% |#####| Time: 0:00:00 248.25 kB/s
llvmlite-0.16. 10% |###| Time: 0:00:03 241.12 kB/s
```

Python

- ▶ Erfunden von Guido van Rossum (Niederlande)
- ▶ Fokus auf lesbaren und verständlichen Code
- ▶ umfangreiche Standard-Bibliothek \Rightarrow „batteries included“
- ▶ Mein erster Kontakt mit Python: Downloadskript für den Save.TV Online-VCR
- ▶ Python2 versus Python3 \Rightarrow Python3
- ▶ Editor/IDE? Ich nutze Spyder3, auch empfehlenswert: Geany
- ▶ Python selbst liefert IDLE mit

Aufgabe 2

Python „Hello World“

- ▶ Editor der Wahl starten, folgende Befehle ausprobieren

```
1 print('Hello Python')
2 # Kommentar
3 a = 123.4
4 a+=2
5 print(a+2)
6
7 def myFunction(a):
8     b = a + a
9     return b
10
11 print(myFunction(2)) # 4
12 print(myFunction('a')) # 'aa'
```

Listing 1: Hello World in Python 3.x, sources/helloWorld.py

Strings, Listen und Tupel

Strings

```
1 a = 'Hallo'
2 b = 'Welt'
3
4 c = a + ' ' + b
5 'W' in c # True
6 print(c[0]) # 'H'
7 print(c[-1]) # 't'
8 print(c[1:3]) # 'al'
9 print(c[1:4]) # 'all'
10 print(c[1:-1]) # 'allo Wel'
11 print(c[1:]) # 'allo Welt'
12
13 for i in c:
14     print(i)
```

Listing 2: Strings, sources/Strings.py

Strings, Listen und Tupel

Stringfunktionen

```
1 meinString = 'Hallo Welt'
2
3 print(meinString.upper())
4 print(meinString.lower())
5 print(meinString.find('W'))
6 print(meinString.split(' '))
7 print(meinString.strip())
8 print(meinString.replace('Welt', 'World'))
```

Listing 3: Strings, sources/Strings2.py

Strings, Listen und Tupel

Listen

- ▶ Index von 0 bis $n - 1$
- ▶ veränderbar

```
1 beatles = ['John', 'Paul', 'Ringo', 'George']
2 print(len(beatles))
3 print(beatles[3])
4 beatles.append('Yoko Ono')
5 print(beatles.index('John'))
```

Listing 4: Listen, sources/Listen.py

Strings, Listen und Tupel

Tupel

- ▶ ähnlich wie Listen
- ▶ Index von 0 bis $n - 1$
- ▶ nicht veränderbar, also „schreibgeschützt“

```
1 monate=('Jan', 'Feb', 'Mar', 'Apr', 'Mai')
2 print(monate[1])
3 print(monate[1:3])
```

Listing 5: Tupel, sources/Tupel.py

Dictionaries

Key-Value Paare

- ▶ nicht veränderbar

```
1 lookup={'EUR': 'Euro', 'GBP': 'Pound', 'USD': 'US-Dollar'}  
2 print(lookup['EUR'])
```

Listing 6: Tupel, sources/Dictionaries.py

Flusssteuerung

if/then

```
1 a = 1
2
3 def doThis():
4     print('Done')
5
6 if a==1:
7     doThis()
8 else:
9     pass # pass = "Do nothing"
```

Listing 7: if-then, sources/ifthen.py

"condition" kann ein üblicher Boolean Ausdruck sein.

Flusssteuerung

for

```
1 myString = 'Python'
2 for c in myString:
3     print(c)
```

Listing 8: for-Schleifen, sources/for.py

Flusssteuerung

while

```
1 n = 4
2 while n>0:
3     print(n)
4     n-=1
```

Listing 9: while-Schleifen, sources/while.py

Funktionen

```
1 def add(a,b):  
2     return a+b  
3  
4 def multiply(a=2,b=3,c=4):  
5     return a*b*c  
6  
7 print(multiply())  
8 print(multiply(2, 4, 6))  
9 print(multiply(a=5))
```

Listing 10: Definition von Funktionen, sources/Funktionen.py

Ein- und Ausgabe

Kommandozeile

```
1 a = input('Erstes Wort')
2 b = input('Zweites Wort')
3
4 print(a, b)
5 print(a, b, sep='')
6 print(a, b, sep=':')
```

Listing 11: Ein- und Ausgabe: Kommandozeile, sources/io.py

Ein- und Ausgabe

Dateien lesen

```
1  dateiname = 'exampleFile.txt'
2
3  filepointerW = open(dateiname, 'w')
4  # 'r' (read) oder 'a' (append)
5  for i in 'Hello World':
6      filepointerW.write(i + '\n')
7  filepointerW.close()
8
9  filepointerR = open(dateiname, 'r')
10 # 'r' (read) oder 'a' (append)
11 for zeile in filepointerR:
12     print(zeile)
13
14 filepointerR.close()
```

Listing 12: Ein- und Ausgabe: Dateien, sources/readWriteFile.py

Ein- und Ausgabe

UTF8-Dateien lesen und schreiben

```
1 import io
2 import datetime
3
4 jetzt = datetime.datetime.now()
5 dateiname = 'example.txt'
6
7 with io.open(dateiname, 'w', encoding='utf8') as datei:
8     datei.write('äüöß ' + jetzt.isoformat())
9
10 with io.open(dateiname, 'r', encoding='utf8') as datei:
11     text = datei.read()
12     print(text)
```

Listing 13: UTF8, sources/readWriteUTF8.py

Andere wichtige Befehle

Das os Modul

os enthält verschiedenste Funktionen für den Umgang mit Dateien und Prozessen, für uns spannend:

- ▶ `os.system(<Befehl>)` Führt externen Befehl aus
- ▶ `os.startfile(<Datei>)` Öffnet Datei mit der Standard-Applikation (nur unter Windows)

Aufgabe 2

Wir (lassen) rechnen...

- ▶ Wir erzeugen Rechenaufgaben für Kinder
- ▶ links die Aufgabe, rechts die Lösung
- ▶ Aufgaben randomisiert (random Modul)
- ▶ Datei automatisch übersetzen und ausführen
- ▶ Tipp für die Schriftart: TeX Gyre Schola,
`\usepackage{tgsschola}`

Lösung

Teil 1

```
1 import random
2 import os
3
4 min = 1 # number to start with
5 max = 100 # maximum number
6 count = 12 # 12 fit on one page
```

Listing 14: Rechnen Teil 1, sources/Aufgaben.py

Lösung

Teil 2

```
1 with open('Aufgaben.tex', 'w') as texfile:
2     texfile.write("\\documentclass[15pt]{scrartcl}\n")
3     texfile.write("\\usepackage[utf8]{inputenc}\n")
4     texfile.write("\\usepackage{tgschola,longtable}\n")
5     texfile.write("\\usepackage[T1]{fontenc}\n")
6     texfile.write("\\setlength{\\parindent}{0pt}\n")
7     texfile.write("\\pagestyle{empty}\n")
8     texfile.write("\\renewcommand*{\\arraystretch}{1.5}\n")
9     texfile.write("\\begin{document}\n")
10    texfile.write("\\huge\n")
11    texfile.write("\\begin{longtable}{cccp{8cm}r}\n")
```

Listing 15: Rechnen Teil 2, sources/Aufgaben.py

Lösung

Teil 3

```
1  for i in range(count):
2      a = random.randint(min, max)
3      b = random.randint(min, max)
4
5      result = a + b;
6
7      texfile.write('%s &+& %s &=& %s \\\n \\hline\n' % (str
          (a), str(b), str(result)))
8
9      texfile.write("\\end{longtable}\n")
10     texfile.write("\\end{document}\n")
11
12 os.system("pdflatex Aufgaben.tex")
13 os.startfile("Aufgaben.pdf") # Windows-only
```

Listing 16: Rechnen Teil 3, sources/Aufgaben.py

Python und \LaTeX verbinden

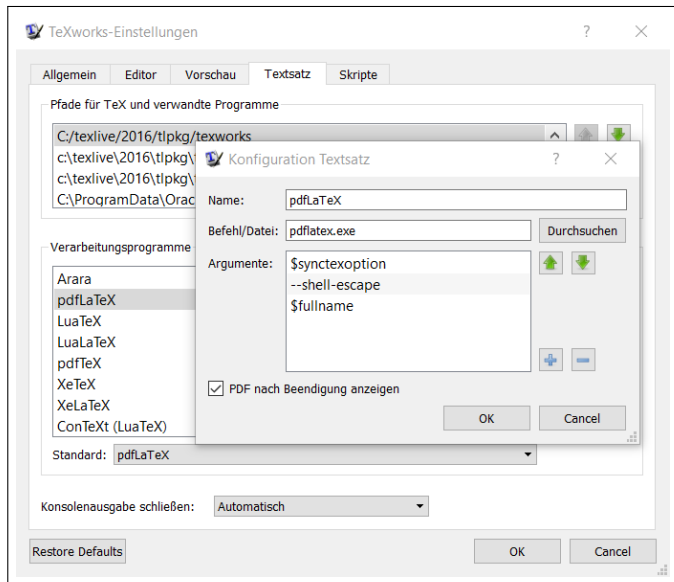
Q: Wie kann man \LaTeX und Python in nur einem Dokument verwalten?

A: Literate programming¹

- ▶ etwas „Selbstgestricktes“
- ▶ Python \TeX von Geoffrey M. Poore
<https://www.ctan.org/pkg/pythontex>

¹WEB, CWEB, NOWEB, SWEAVE, knitr

Python im L^AT_EX-Lauf



Python im L^AT_EX-Lauf

```
1 \makeatletter
2 \newenvironment{pycode}[1]%
3   {\xdef\d@tn@me{#1}\xdef\r@ncmd{python #1.py > #1.plog}%
4   \typeout{Writing file #1}\VerbatimOut{#1.py}%
5   }
6   {\endVerbatimOut %
7   \toks0{\immediate\write18}%
8   \expandafter\toks\expandafter1\expandafter{\r@ncmd}%
9   \edef\d@r@ncmd{\the\toks0{\the\toks1}}\d@r@ncmd %
10  \lstinputlisting[language={Python},label=listing:\d@tn@me
11    ,basicstyle={\ttfamily\footnotesize}]{\d@tn@me.py}%
12  \lstinputlisting[basicstyle={\ttfamily\footnotesize}]{\d@tn@me.plog}%
13 }
```

Python im L^AT_EX-Lauf

Beispiele unter /sources, leicht anpassbar

`pycode-mini.tex` einspaltig s/w

`pycode.tex` einspaltig, farbig hinterlegt

`pycode-beamer.tex` für Folien, zweispaltig

PythonT_EX

- ▶ „PythonTeX“ Paket von Geoffrey M. Poore
- ▶ Paket in Version 0.15 auf CTAN, auch in T_EX Live 2016 enthalten
- ▶ unter Windows `pythontex.exe`
- ▶ stellt mehrere Befehle und Umgebungen bereit
- ▶ Syntax-Highlighting via `pygments`
- ▶ komplexer als die „Eigenbau-Lösung“
- ▶ Workflow
 - ▶ Kompiliere Dokument mit `*latex`
 - ▶ Lass `pythontex.exe/pythontex.py` laufen
 - ▶ Kompiliere Dokument wieder mit `*latex`

Arara-Regel

Hilfreich dabei: eigene Arara-Regel

- ▶ Was ist Arara?
- ▶ texwelt.de/wissen/fragen/8764/was-ist-arara
- ▶ ablegen unter C:/texlive/2016/texmf-dist/scripts/arara/rules oder im lokalen texmf-Tree

```
1 !config
2 # Nomentbl rule for arara
3 # author: Uwe Ziegenhagen
4 # requires arara 3.0+
5 identifier: pythontex
6 name: pythontex
7 command: <arara> pythontex @{{options}} "@{{getBasename(file)}}.pytxcode"
8 arguments:
9 - identifier: style
10   flag: <arara> @{{parameters.style}}
11   default: pythontex
12 - identifier: options
13   flag: <arara> @{{parameters.options}}
```

Listing 17: UTF8, sources/pythontex.yaml

PythonT_EX

Befehle

- ▶ `\py{}` für Code, der einen String zurückliefert
- ▶ `\pyc{}` für Code, der nur ausgeführt wird
- ▶ `\pys{}` mit Substitution
- ▶ `\pyv{}` für Code, der nur gesetzt wird
- ▶ `\pyb{}` für Ausführung und Satz der Ergebnisse

Umgebungen

`pycode` wie `pyc`

`pysub` wie `pys`

`pyverbatim` wie `pyv`

`pyblock` wie `pyb`

`pyconsole` simuliert eine Python-Konsole

PythonT_EX

Beispiele aus der PythonT_EX-Galerie

Alle Beispiele unter /sources

[pythontex-01.tex](#) „Hallo Welt“

[pythontex-02.tex](#) Formatstring

[pythontex-03.tex](#) Substitution

[pythontex-04.tex](#) Unicode und Pyconsole

[pythontex-05.tex](#) Pyconsole Evaluation

[pythontex-06.tex](#) SymPy

Eigenes Beispiel

Erzeugung einer Verteilungstabelle

```
1 from scipy.stats import norm
2
3 print('\begin{tabular}{r|cccccccc} \toprule')
4
5 horizontal = range(0,10,1)
6 vertikal = range(0,37)
7
8 header = ''
9 for i in horizontal :
10     header = header + ' & ' + str(i/100)
11
12 print(header, '\\\\ \midrule')
13
14 for j in vertikal :
15     x = j/10
16     print('\\\\', x)
17     for i in horizontal :
18         y = x + i/100
19         print('& ', "{:10.4f}".format(norm.cdf(y)))
```

Listing 18: Normalverteilung, sources/pythontex-07.tex

Daten verarbeiten

Die wunderbare Welt von pandas

- ▶ mein „täglich Brot“: Datenbestände zwischen Banksystemen abgleichen
- ▶ „pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.“²
- ▶ Initiiert 2008 durch Wes McKinney von AQR Capital Management für hoch-performante quantitative Analyse
- ▶ Wesentliche Teile in C/Cython implementiert
- ▶ Current version is 0.19

²Source: pandas.pydata.org

Series und DataFrames

- ▶ central data structures in pandas

		Column Index						
		'var 0'	'var 1'	'var 2'	'var 3'	'var 4'	'var 5'	'var 6'
Row Index	0	0.2	'USD'	...				
	1	0.4	'EUR'	...				
	2	0.1	'USD'	...				
	3	0.7	'EUR'	...				
	4	0.5	'YEN'	...				
	5	0.5	'USD'	...				
	6	0.0	'AUD'	...				

Daten lesen

Command	Description
<code>read_pickle</code>	lies Pickle objects
<code>read_table</code>	für Tabellenformate
<code>read_csv</code>	Comma-Separated Values
<code>read_fwf</code>	für fixed-width Formate
<code>read_clipboard</code>	lies aus der Zwischenablage
<code>read_excel</code>	lies Excel-Dateien

andere Befehle für HTML, JSON, HDF5, ...

CSV-Formate lesen

- ▶ CSV \neq CSV
 - ▶ Spaltentrenner
 - ▶ Dezimaltrenner
 - ▶ Encoding
- ▶ http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html
 - `sep` Spaltentrenner
 - `thousands` Tausender-Trenner
 - `encoding` hoffentlich UTF8
 - `decimal` Dezimaltrenner
 - `converters` `converters={'A': str}` für Konvertierung

Excel lesen

- ▶ `pd.read_excel()` für XLSX-Dateien
- ▶ Dokumentation:
`http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_excel.html`
- ▶ Export nach Excel: `pd.to_excel()` function
- ▶ Hinweise:
 - ▶ Excel-export langsamer als CSV-Export
 - ▶ „Schickes“ Excel ist aufwändiger
 - ▶ Excel/Office kann auch gut per COM gesteuert werden

DataFrames abfragen

Grundlegende Daten

```
1 customers = pd.read_excel('Northwind.xlsx',  
2                             sheetname = 'Customers')  
3  
4 print(len(customers)) # Zeilenzahl  
5 print(customers.head()) # die ersten 5 Zeilen  
6 print(customers.tail()) # die letzten 5 Zeilen  
7 print(list(customers)) # die Spaltennamen
```

Pandas Dataframe Operationen

Auswahl und Filterung I

- ▶ pandas hat clevere Funktionen zur Datenauswahl
- ▶ Wähle bestimmte Spalten aus
`df = df[['colA', 'colB']]`
- ▶ Wähle nur die ersten zwei Zeilen (Index beginnt mit 0)
`df.iloc[:1]`
- ▶ Wähle die Zeilen, in denen `colA > 50`
`df[df['colA'] > 50]`

Pandas Dataframe Operationen

Auswahl und Filterung II

- ▶ Wähle die Zeilen, in denen colA größer 50 und kleiner 500
`df[(df['colA'] > 50) | (df['colA'] < 500)]`
- ▶ Wähle die Zeilen, in denen colA nicht „HelloWorld“ ist
`df[~(df['colA'] == 'HelloWorld')]`
- ▶ Wähle die Zeilen, in denen 'b' 'A' oder 'I' ist
`df = df[(df['b'] == 'A') | (df['b'] == 'I')]`
- ▶ Alternative dazu via `isin()`
`df = df[df['b'].isin(['A', 'I'])]`
- ▶ oder das Gegenstück dazu
`df = df[~df['b'].isin(['A', 'I'])]`

Pandas Dataframe Operationen

Merge

- ▶ `merge()` join wie in SQL
- ▶ nützlich, um Datensätze zu verbinden
- ▶ Unterstützt werden
 - ▶ Left
 - ▶ Right
 - ▶ Inner
 - ▶ Full Outer

Pandas Dataframe Operationen

Merging

► Optionen für merge()

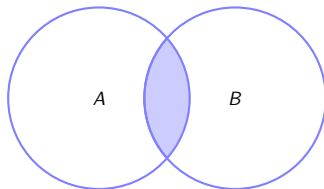
```
1 leftDataFrame.merge(rightDataFrame, how='inner',  
2 on=None, left_on=None, right_on=None, left_index=False,  
3 right_index=False, sort=False, suffixes=('_x', '_y'),  
4 copy=True, indicator=False)
```

1. Definiere den anderen Datensatz
2. Definiere, wie verbunden werden soll
3. Definiere die Schlüsselspalten

Merging

Inner Join

- ▶ Alle Daten, die in A und B sind



left		
	A	Key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K4

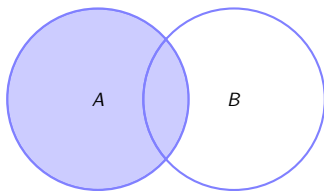
right		
	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

merged			
	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2

Merging

Left Join

- ▶ Alle Daten, die in A sind, mit passenden Daten aus B



left		
	A	Key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K4

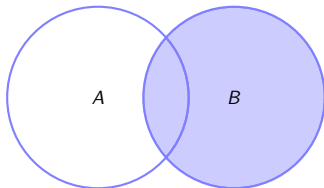
right		
	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

merged			
	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	NaN	K4

Merging

Right Join

- ▶ Alle Daten, die in B sind, mit passenden Daten aus A



left		
	A	Key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K4

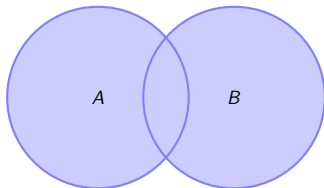
right		
	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

merged			
	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	NaN	B3	K5

Merging

Full Outer Join

- ▶ Alle Daten, die in A oder B sind



left		
	A	Key
0	A0	K0
1	A1	K1
2	A2	K2
3	A3	K4

right		
	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

merged			
	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	NaN	K4
4	NaN	B3	K5

jinja2

Was ist eine „Template Engine“?

Jinja2 is a modern and designer-friendly templating language for Python, modelled after Django's templates. It is fast, widely used and secure with the optional sandboxed template execution environment.

- ▶ Vorlagen (aus Dateien, Datenbank, etc.)
- ▶ Liste von Variablen
- ▶ jinja2 ersetzt die Variablen durch Inhalte
- ▶ Erlaubt Trennung von Programmcode und Vorlage

jinja2

Ein einfaches Beispiel (ohne \LaTeX)

```
1 Hallo {{variable}}
```

Listing 19: Inhalt der Datei „test-min.txt“

```
1 import jinja2
2 import os
3 from jinja2 import Template
4
5 jinja_env = jinja2.Environment(
6     loader = jinja2.FileSystemLoader(os.path.abspath('.'))
7 )
8
9 template = jinja_env.get_template('test-min.txt')
10 print(template.render(variable='Welt'))
```

Listing 20: UTF8, sources/jinja2-01.py

jinja2

Eingebaute Schleifen

```
1 from jinja2 import Template
2
3 itemlist = ['first','second','third']
4
5 template = Template(
6     '''\begin{itemize}
7     {% for item in liste %}
8         \item {{item}}
9     {% endfor %}
10    \end{itemize}'''
11 )
12
13 print(template.render(liste=itemlist))
```

Listing 21: UTF8, sources/jinja2-02.py

Beispiel: Erstellung von Spendenbescheinigungen

- ▶ Schatzmeister des Kölner Dingfabrik e.V.
- ▶ Manuelle Erstellung der Spendenbescheinigungen keine Option
- ▶ Bis 2014: Python, MySQL, etc. (Siehe meinen Vortrag 2014 in Heidelberg)
- ▶ Seither pandas, deutlich einfacher
- ▶ Mehr dazu unter <http://uweziegenhagen.de/?p=3359>

Schauen wir uns das Beispiel unter
`sources/Spendenquittungen` an