

Diejenigen, die aktiv mitmachen wollen, setzen sich bitte nach vorn.

# Python & L<sup>A</sup>T<sub>E</sub>X

Dante e. V. Frühjahrstagung 2017

Dr. Uwe Ziegenhagen

22. März 2017

# Überblick

Was machen wir heute?

- ▶ Python Grundlagen
- ▶ Python in  $\text{\LaTeX}$  Dokumenten
- ▶ Erzeugung von  $\text{\LaTeX}$  Dokumenten

# Voraussetzungen

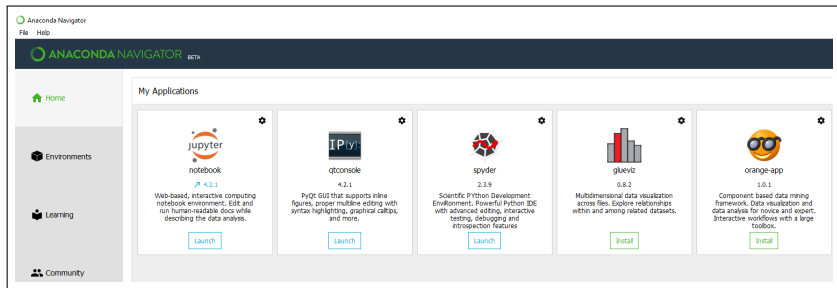
## Was wird benötigt

- ▶ Aktuelle T<sub>E</sub>X Installation
- ▶ Python3 Installation, vorzugsweise Anaconda/Winpython
- ▶ 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

Neben pandas ist enthalten:

**NumPy** matrices, vectors, algorithms

**IPython** Matlab/Mathematica-like environment

**Matplotlib** scientific plotting, basis for seaborn library

**SymPy** symbolic mathematics

... etc, etc.

# Testen der Installation

- Funktionieren die folgenden Befehle?

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

# Python

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



# Python „Hello World“

```
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 meinString.upper()
4 meinString.find('Welt')
5 meinString.split(' ')
6 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/Dictionaryes.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 multiply3(a,b,c=1):  
5     return a*b*c
```

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 filepointer = open(dateiname,'r')
4 # 'r' (read) oder 'a' (append)
5 for zeile in filepointer:
6     print(zeile)
7
8 filepointer.close()
```

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

- ▶ Um Excel, CSV und ähnliches zu lesen ⇒ pandas sehr empfehlenswert

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

# Aufgabe 1

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

# Python und $\text{\LaTeX}$ verbinden

Q: Wie kann man  $\text{\LaTeX}$  und Python in nur einem Dokument verwalten?

A: Literate programming<sup>1</sup>

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

---

<sup>1</sup>WEB, CWEB, NOWEB, SWEAVE, knitr



# Python im L<sup>A</sup>T<sub>E</sub>X-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 }
14 \makeatother
```

- Beispiele unter /sources, leicht anpassbar

# PythonT<sub>E</sub>X

- ▶ „PythonTeX“ Paket von Geoffrey M. Poore
- ▶ Paket in Version 0.15 auf CTAN, in T<sub>E</sub>X Live enthalten
- ▶ unter Windows `pythontex.exe`
- ▶ stellt mehrere Befehle und Umgebungen bereit

- ▶ Kompiliere Dokument mit `*latex`
- ▶ Lass `pythontex.exe` laufen
- ▶ Kompiliere Dokument wieder mit `*latex`

# Arara-Regel

- ▶ Was ist Arara?
- ▶ [texwelt.de/wissen/fragen/8764/was-ist-arara](http://texwelt.de/wissen/fragen/8764/was-ist-arara)

```
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 14: UTF8, sources/pythontex.yaml

# PythonT<sub>E</sub>X

## 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

# 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.“<sup>2</sup>
- ▶ 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

---

<sup>2</sup>Source: [pandas.pydata.org](http://pandas.pydata.org)

# Series und DataFrames

- ▶ central data structures in pandas

A diagram illustrating a pandas DataFrame structure. It consists of a grid of cells. The first column contains row indices from 0 to 6, highlighted in light blue. A red arrow on the left points downwards, labeled 'Row Index'. The first row contains column labels: 'var 0', 'var 1', 'var 2', 'var 3', 'var 4', 'var 5', and 'var 6', highlighted in yellow. A red arrow at the top points to the right, labeled 'Column Index'. The data cells are white with black borders. The values in the first column are 0, 1, 2, 3, 4, 5, and 6. The values in the second column are 0.2, 0.4, 0.1, 0.7, 0.5, 0.5, and 0.0. The values in the third column are 'USD', 'EUR', 'USD', 'EUR', 'YEN', 'USD', and 'AUD'. The values in the fourth column are '...', '...', '...', '...', '...', '...', and '...'. The remaining cells in the first four columns are empty.

	'var 0'	'var 1'	'var 2'	'var 3'	'var 4'	'var 5'	'var 6'
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](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()

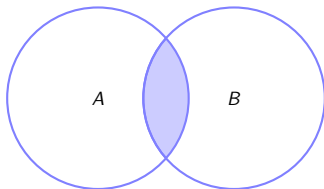
```
1 leftDataFrame.merge(rightDataFrame, how='inner',  
2 on=None, left_on=None, right_on=None, left_index=False,  
3 right_index=False, sort=False, suffixes=('_x', '_y'),  
4 copy=True, indicator=False)
```

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

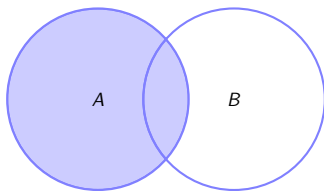
merged			
	A	B	Key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2



# 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

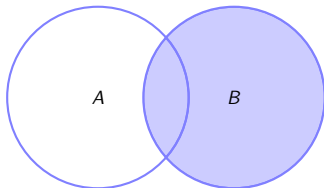
right		
	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

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

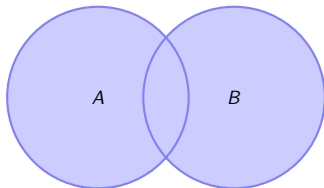
right		
	B	Key
0	B0	K0
1	B1	K1
2	B2	K2
3	C3	K5

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

# jinja2

Ein einfaches Beispiel (ohne  $\text{\LaTeX}$ )

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

Listing 15: 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 16: 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 17: 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 meine Vortrag 2014 in Heidelberg)
- ▶ Seither pandas, deutlich einfacher
- ▶ Mehr dazu unter <http://uweziegenhagen.de/?p=3359>