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

Python & LaTeX

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

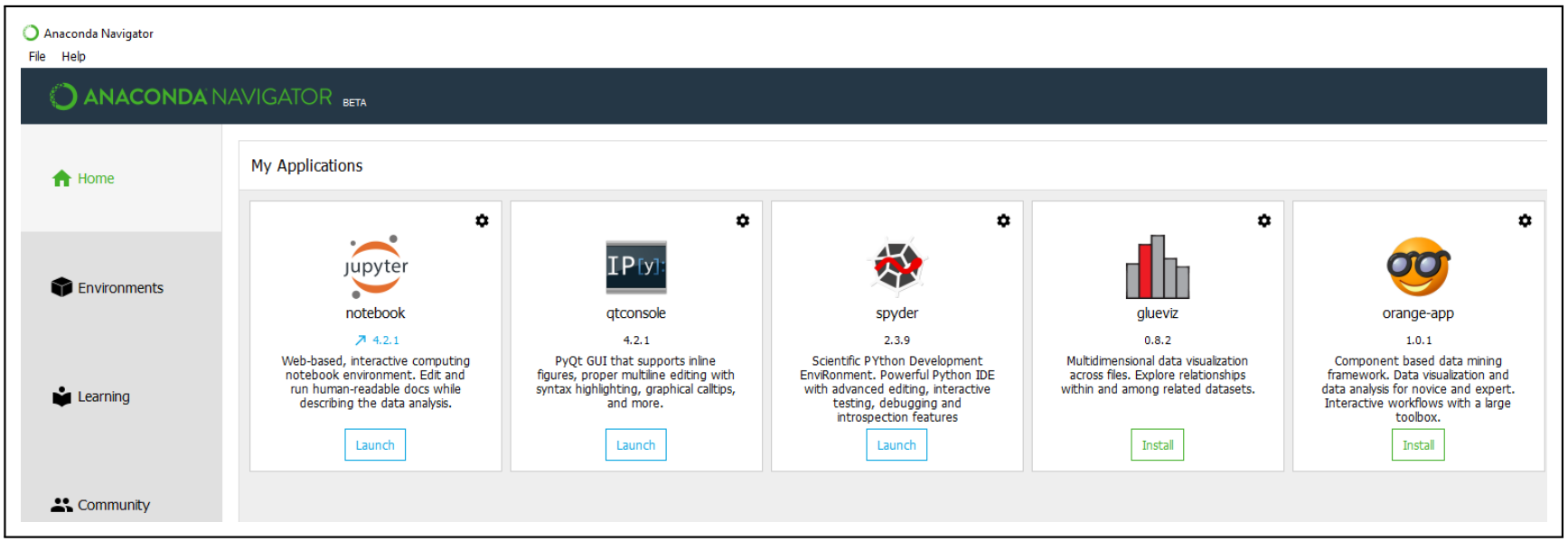
Was wird benötigt

- Aktuelle TEX 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/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.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 LaTeX verbinden

Q: Wie kann man LaTeX und Python in nur einem Dokument verwalten?
A: Literate programming[1]

- etwas „Selbstgestricktes“
- PythonTeX von Geoffrey M. Poore
  `https://www.ctan.org/pkg/pythontex`

[1] WEB, CWEB, NOWEB, SWEAVE, knitR

Python im LaTeX-Lauf

```
\makeatletter
\newenvironment{pycode}[1]%
  {\xdef\d@tn@me{#1}\xdef\r@ncmd{python #1.py > #1.plog}%
  \typeout{Writing file #1}\VerbatimOut{#1.py}%
  }
  {\endVerbatimOut %
 \toks0{\immediate\write18}%
 \expandafter\toks\expandafter1\expandafter{\r@ncmd}%
 \edef\d@r@ncmd{\the\toks0{\the\toks1}}\d@r@ncmd %
 \lstinputlisting[language={Python},label=listing:\d@tn@me
     ,basicstyle={\ttfamily\footnotesize}]{\d@tn@me.py}%
 \lstinputlisting[basicstyle={\ttfamily\footnotesize}]{\
     d@tn@me.plog}%
}
\makeatother
```

- Beispiele unter /sources, leicht anpassbar

PythonTEX

- „PythonTeX“ Paket von Geoffrey M. Poore
- Paket in Version 0.15 auf CTAN, in TEX Live enthalten
- unter Windows `pythontex.exe`
- stellt mehrere Befehle und Umgebungen bereit

PythonTEX

Arbeitsweise

- 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

```
!config
# Nomentbl rule for arara
# author: Uwe Ziegenhagen
# requires arara 3.0+
identifier: pythontex
name: pythontex
command: <arara> pythontex @{options} "@{getBasename(file)}.pytxcode"
arguments:
- identifier: style
  flag: <arara> @{parameters.style}
  default: pythontex
- identifier: options
  flag: <arara> @{parameters.options}
```

Listing 14: UTF8, sources/pythontex.yaml

PythonTEX

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.“[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

[2] Source: pandas.pydata.org

Series und DataFrames

- central data structures in `pandas`

Daten lesen

Command	Description
read_pickle	lies Pickle objects
read_table	für Tabellenformate
read_csv	Comma-Separated Values
read_fwf	für fixed-width Formate
read_clipboard	lies aus der Zwischenablage
read_excel	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

```
customers = pd.read_excel('Northwind.xlsx',
                          sheetname = 'Customers')

print(len(customers))   # Zeilenzahl
print(customers.head()) # die ersten 5 Zeilen
print(customers.tail()) # die letzten 5 Zeilen
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 'B' 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()`

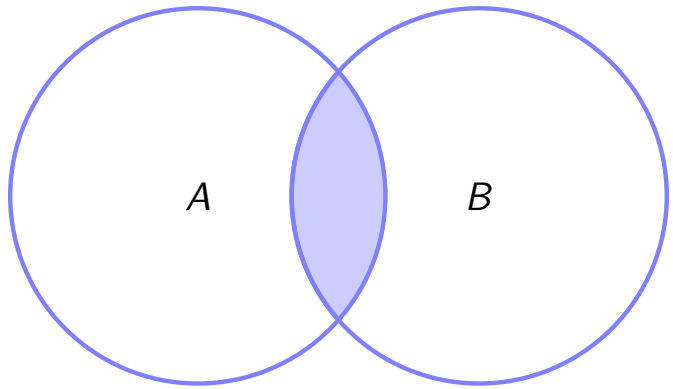
```
leftDataFrame.merge(rightDataFrame, how='inner',
on=None, left_on=None, right_on=None, left_index=False,
right_index=False, sort=False, suffixes=('_x', '_y'),
copy=True, indicator=False)
```

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

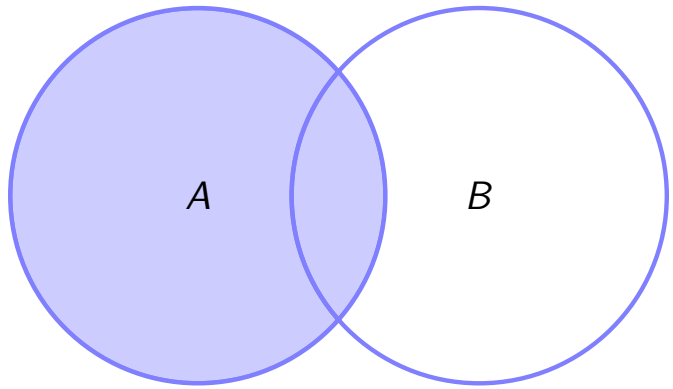
merged

	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

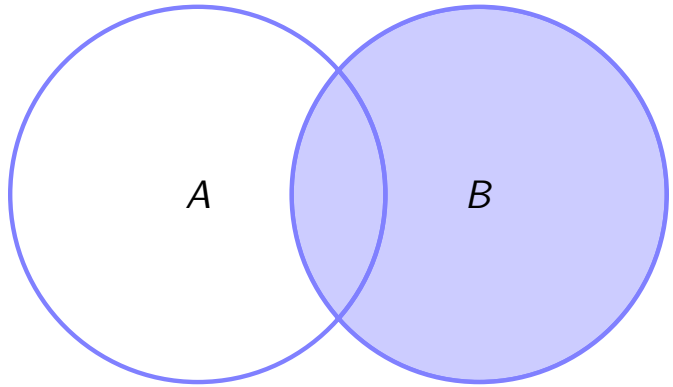
merged

	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

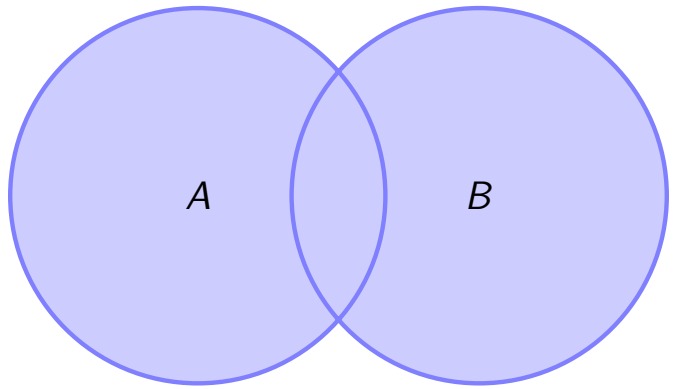
merged

	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 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`