Carsten Knoll

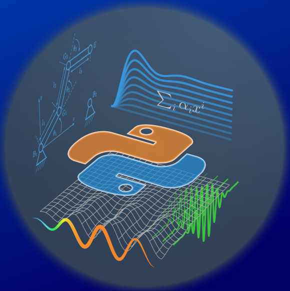
Chair of Fundamentals of Electrical Engineering

Python for Engineers
Pythonkurs für Ingenieur:innen

Organisation and Python basics
Organisatorisches und Python-Grundlagen

Dresden (online), 2023-10-10

https://tu-dresden.de/pythonkurs
https://python-fuer-ingenieure.de



Sprache / Language

This year the course is part of the module
Neural Networks and Memristive Hardware Accelerators.
Module is in english ⇒ course bilingual.

If there is a language barrier: **please ask!**

Wenn Sie etwas nicht verstehen: **Bitte fragen!**

Self Introduction: Carsten Knoll

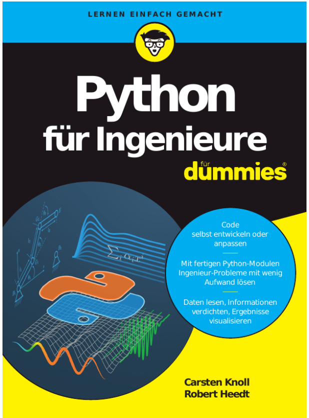
- Postdoc at Chair of Fundamentals of Electrical Engineering
  - Research: Explainable AI, Semenatic Technology, Control Theory
- Co-Founder of:
  - Hochschulgruppe für Freie Software und Freies Wissen
  - Bits und Bäume Dresden

  → Interested people are always welcome (ask me!)

FSFW

- First experience with Python in 2004, active usage since 2008

Book (currently German only)



LERNEN EINFACH GEMACHT

Python für Ingenieure

für dummies®

Code selbst entwickeln oder anpassen

Mit fertigen Python-Modulen Ingenieur-Probleme mit wenig Aufwand lösen

Daten lesen, Informationen verdichten, Ergebnisse visualisieren

Carsten Knoll
Robert Heedt

https://python-fuer-ingenieure.de

(Every code example from the code is available. One Notebook per chapter!)

History of the course

- Python usage of individual people → contact, exchange.
- Observation: lack of Python courses

→ Summer Semester 2011: volunteer initiative: organization of a course

History of the course

- Python usage of individual people → contact, exchange.
- Observation: lack of Python courses

→ Summer Semester 2011: volunteer initiative: organization of a course

- 2013/14: Funding by the “Hochschuldidaktisches Zentrum Sachsen” (HDS)

→ Personnel funds for the implementation of the "‘Blended Learning"’ concept: Self-learning phases- and online consultations in alternation

  - Self-learning phases: Knowledge transfer (screencasts), comprehension quizzes, exercises (to get an overview)
  - Online consultations: Complex exercises (in groups, with opportunity to ask questions)

History of the course

- Python usage of individual people → contact, exchange.
- Observation: lack of Python courses

→ Summer Semester 2011: volunteer initiative: organization of a course

- 2013/14: Funding by the “Hochschuldidaktisches Zentrum Sachsen” (HDS)

→ Personnel funds for the implementation of the "“Blended Learning"” concept: Self-learning phases- and online consultations in alternation

  - Self-learning phases: Knowledge transfer (screencasts), comprehension quizzes, exercises (to get an overview)
  - Online consultations: Complex exercises (in groups, with opportunity to ask questions)

- Different course names over time
- Since WiSe 17/18 gender-neutral title (in German) "“... for engineers"” (background)
- Since 2020/21 completely online

Former Co-Creators

- Dipl.-Ing. Sebastian Voigt
- Dipl.-Ing. Christoph Statz
- Dipl.-Inf. Ingo Keller
- Dipl.-Ing. Peter Seifert

- Dr.-Ing. Ines Gubsch
- Andreas Kunze
- Dominik Pataky
- Victoria Vinis

- Many thanks to
  - M. Grabowski, C. J. Kleine (ZIH), ...

Why Python? (1)

Python as Programming language

- Clear, readable syntax (little "“overhead"")
- Object-oriented, procedural, functionally programmable
- Useful built-in data types (`list, tuple, dict, set, ...`)
- Easy modularization (`import this`)
- Good error management (exceptions)
- Extensive standard library
- Easy integration of external code (C, C++, Fortran)

Why Python? (1)

Python as Programming language

- Clear, readable syntax (little "“overhead"”)
- Object-oriented, procedural, functionally programmable
- Useful built-in data types (`list, tuple, dict, set, ...`)
- Easy modularization (`import this`)
- Good error management (exceptions)
- Extensive standard library
- Easy integration of external code (C, C++, Fortran)

⇒

- Easy to learn
- Problem oriented (powerful and flexible)
- Motivation potential ↗, frustration potential ↘

Also: cross-platform / free and open source / large & active community

Why Python? (2)

Python as a tool for engineers:

- Symbolic calculation (derive, integrate, solve equations, ...)
- Numerical calculation (lin. algebra, DGLn, optimization, ...)
- Visualization (2D, 3D, in publication quality)
- Graphical User Interface (GUI)
- Parallelization

- Communication with external devices (RS232, GPIB, ...)

Why Python? (2)

Python as a tool for engineers:

- Symbolic calculation (derive, integrate, solve equations, ...)
- Numerical calculation (lin. algebra, DGLn, optimization, ...)
- Visualization (2D, 3D, in publication quality)
- Graphical User Interface (GUI)
- Parallelization

} prospective course content

- Communication with external devices (RS232, GPIB, ...)

- Python useful for other subjects/projects

⇒ Strengthened "“research competence"” (study and master/diploma theses, ...)

Didactic concept: Programming is learned by *actively* reading and writing code

Self-learning phases (60-90min per week)

- Opal: teaching material + further links
- Overview lecture as screencast (basic concepts, commands, traps) "‘finger exercises"’ → type along yourself (play around)
- CodeQuiz: Comprehension questions
- Look at exercises, understand the task, compare with lecture material, try to solve

Didactic concept: Programming is learned by *actively* reading and writing code

Self-learning phases (60-90min per week)

- Opal: teaching material + further links
- Overview lecture as screencast (basic concepts, commands, traps) "“finger exercises"” → type along yourself (play around)
- CodeQuiz: Comprehension questions
- Look at exercises, understand the task, compare with lecture material, try to solve

Online consultations (see also `https://yopad.eu/p/tud-pythonkurs-365days`)

- Complex exercise task
  - Subdivided into manageable subtasks.
  - Provided: source code fragments + detailed solution
- Programming in groups ("pair programming")
  - Enables and requires cooperation

Didactic concept: Programming is learned by *actively* reading and writing code

Self-learning phases (60-90min per week)

- Opal: teaching material + further links
- Overview lecture as screencast (basic concepts, commands, traps) "“finger exercises"” → type along yourself (play around)
- CodeQuiz: Comprehension questions
- Look at exercises, understand the task, compare with lecture material, try to solve

Online consultations (see also `https://yopad.eu/p/tud-pythonkurs-365days`)

- Complex exercise task
  - Subdivided into manageable subtasks.
  - Provided: source code fragments + detailed solution
- Programming in groups ("pair programming")
  - Enables and requires cooperation

Programming project (dt: „Beleg“) (manageable programming task)

- Condition for certificate of achievement (unfortunately without Credit Points!)
- Last event: Presentation and short discussion about the program

→ Your own topic suggestions welcome

Preparation and Installation

- **Recommendation** Anaconda distribution, installed via the miniconda installer: `https://docs.conda.io/en/latest/miniconda.html`
- **Assumption:** You can open a command line window (="‘Console"’ = "‘Terminal"’) with the Conda environment enabled ("‘Anaconda prompt"’):
- Install relevant packages with

```
# - Installation benötigter Zusatzpakete
pip install numpy scipy matplotlib notebook ipywidgets symbtools ipydex
pip install spyder
```

- background information: We use Python >=3.8

Preparation and Installation

- **Recommendation** Anaconda distribution, installed via the miniconda installer: `https://docs.conda.io/en/latest/miniconda.html`
- **Assumption:** You can open a command line window (="‘Console"’ = "‘Terminal"’) with the Conda environment enabled ("‘Anaconda prompt"’):
- Install relevant packages with

```
# - Installation benötigter Zusatzpakete
pip install numpy scipy matplotlib notebook ipywidgets symbtools ipydex
pip install spyder
```

- background information: We use Python >=3.8
- Python 3.x is **not** 100% backward compatible (e. B. `print "hello"` → `print("hello")`)

⚠ You can still find a lot of 2.x code on the net, e.g. the old german screencasts of this course

4 Ways to Start Python

- default interpreter (interactive mode)
  - command: `python` or `python3`
  - can execute any Python command; (might be useful for basic testing or as a calculator)

4 Ways to Start Python

- default interpreter (interactive mode)
  - command: `python` or `python3`
  - can execute any Python command; (might be useful for basic testing or as a calculator)
- IPython (“Enhanced Interactive Python” → **clearly** better)
  - command: `ipython`
  - tab completion, intelligent history, dynamic object info (with `?` or `??`), colored error messages

4 Ways to Start Python

- default interpreter (interactive mode)
  - command: `python` or `python3`
  - can execute any Python command; (might be useful for basic testing or as a calculator)
- IPython ("Enhanced Interactive Python" → **clearly** better)
  - command: `ipython`
  - tab completion, intelligent history, dynamic object info (with `?` or `??`), colored error messages
- Integrated Development Environment (e.g. spyder)
  - command `spyder3`
  - customized text editor
  - Significantly more programming relevant functions

4 Ways to Start Python

- default interpreter (interactive mode)
  - command: `python` or `python3`
  - can execute any Python command; (might be useful for basic testing or as a calculator)
- IPython (“Enhanced Interactive Python” → **clearly** better)
  - command: `ipython`
  - tab completion, intelligent history, dynamic object info (with `?` or `??`), colored error messages
- Integrated Development Environment (e.g. spyder)
  - command `spyder3`
  - customized text editor
  - Significantly more programming relevant functions
- Jupyter Notebook (with Python kernel)
  - command: `cd pykurs-wise2023-24; jupyter notebook .` (change to course directory; start the notebook server there)
  - backend: (local) web server; frontend: Interactive document in browser
  - notebooks combine source code, program output and documentation (incl. LaTeX formulas)

Jupyter

Key keyboard commands

command mode (`Esc` to enable)

- `Shift-Return` - execute cell, activate next one
- `h` - show keyboard commands
- `m` - change cell type to "‘markdown"’
- `y` - change cell type to "‘code"’
- `a` - new cell above
- `b` - new cell below

edit mode (`enter` to activate)

- `Shift-Return` - execute cell, activate next one
- `Tab` - Autocomplete or indentation
- `Shift-Tab` - Remove indentation
- `Ctrl-Z` - Undo

Jupyter

Key keyboard commands

command mode (`Esc` to enable)

- `Shift-Return` - execute cell, activate next one
- `h` - show keyboard commands
- `m` - change cell type to "‘markdown"’
- `y` - change cell type to "‘code"’
- `a` - new cell above
- `b` - new cell below

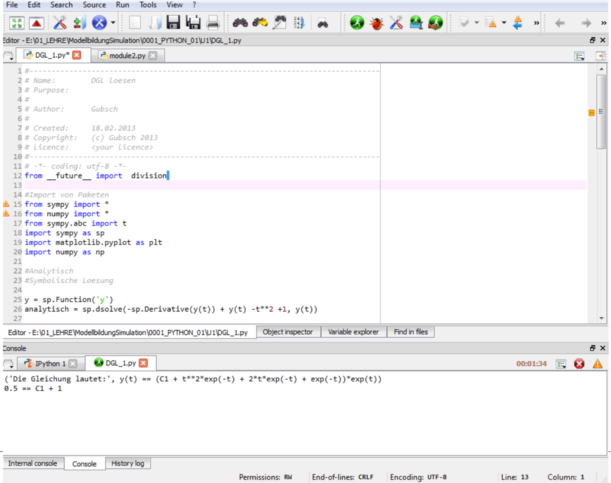
edit mode (`enter` to activate)

- `Shift-Return` - execute cell, activate next one
- `Tab` - Autocomplete or indentation
- `Shift-Tab` - Remove indentation
- `Ctrl-Z` - Undo

→ Start the server in the current directory: `jupyter notebook ./`

→ Trying `example-notebook1.ipynb`

Spyder Integraged Development Environment (IDE)



```
#-------------------------------------------------------------------------------
# Name:        DGL loesen
# Purpose:
#
# Author:      Gubsch
#
# Created:     18.02.2013
# Copyright:   (c) Gubsch 2013
# Licence:     <your licence>
#-------------------------------------------------------------------------------
# -*- coding: utf-8 -*-
from __future__ import  division

#Import von Paketen
from sympy import *
from numpy import *
from sympy.abc import t
import sympy as sp
import matplotlib.pyplot as plt
import numpy as np

#Analytisch
#Symbolische Loesung

y = sp.Function('y')
analytisch = sp.dsolve(-sp.Derivative(y(t)) + y(t) -t**2 +1, y(t))
```

```
('Die Gleichung lautet:', y(t) == (C1 + t**2*exp(-t) + 2*t*exp(-t) + exp(-t))*exp(t))
0.5 == C1 + 1
```

Code Example

Listing: hello-world.py

```python
import math
print("Hello World")
a = 10
b = 20.5
c = a + b + 3**2
print(math.sqrt(c))

while True: # start infinite loop
    x = input("Your name? ")  # returns a str-object
    if x == "q":
        break # finish loop
    print("Hello ", x)
```

Code Example

Listing: hello-world.py

```
import math
print("Hello World")
a = 10
b = 20.5
c = a + b + 3**2
print(math.sqrt(c))

while True:  # start infinite loop
    x = input("Your name? ")  # returns a str-object
    if x == "q":
        break # finish loop
    print("Hello ", x)
```

- Indentations have syntactical meaning!
- de facto standard: 4 spaces. (in editors: Blockwise with <TAB>(→) and <SHIFT+TAB> (←)).

Code Example

Listing: hello-world.py

```
import·math
print("Hello·World")
a·=·10
b·=·20.5
c·=·a·+·b·+·3**2
print(math.sqrt(c))

while·True:·#·start·infinite·loop
····x·=·input("Your·name?·")··#·returns·a·str-object
····if·x·==·"q":
········break·#·finish·loop
····print("Hello·",·x)
```

- Indentations have syntactical meaning!
- de facto standard: 4 spaces. (in editors: Blockwise with <TAB>(→) and <SHIFT+TAB> (←)).
- Python basics: see: `01_overview_Python_types_and_syntax.pdf`

IPython (Interactive Python)

- *Interactive* working very useful when ...
  - “exploring” new modules, features, ...
  - searching for bugs
- IPython = “Interactive Python” = improved Python shell
  - history
  - auto-completion (<TAB>)
  - simple help (<?> and <??>)
  - colors, “magic” commands (e.g. `%time`), ...

IPython (Interactive Python)

- *Interactive* working very useful when …
  - “exploring” new modules, features, …
  - searching for bugs
- IPython = “Interactive Python” = improved Python shell
  - history
  - auto-completion (<TAB>)
  - simple help (<?> and <??>)
  - colors, “magic” commands (e.g. `%time`), …
- can also be embedded in your own programs:

```
from ipydex import IPS
...
x = "abcdefg"
IPS()
...
```

IPython (Interactive Python)

- *Interactive* working very useful when ...
  - “exploring” new modules, features, ...
  - searching for bugs

- IPython = “Interactive Python” = improved Python shell
  - history
  - auto-completion (<TAB>)
  - simple help (<?> and <??>)
  - colors, “magic” commands (e.g. `%time`), ...

- can also be embedded in your own programs:

```python
from ipydex import IPS
...
x = "abcdefg"
IPS()
...
```

- integration in Spyder not so good

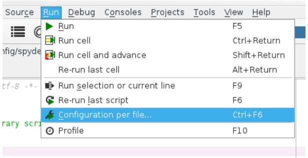
IPython (Interactive Python)

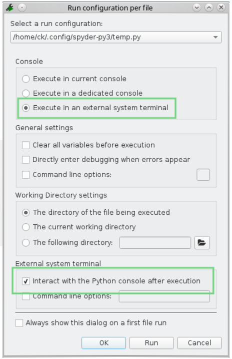
- *Interactive* working very useful when …
  - “exploring” new modules, features, …
  - searching for bugs
- IPython = “Interactive Python” = improved Python shell
  - history
  - auto-completion (<TAB>)
  - simple help (<?> and <??>)
  - colors, “magic” commands (e.g. `%time`), …
- can also be embedded in your own programs:

```
from ipydex import IPS
...
x = "abcdefg"
IPS()
...
```

- integration in Spyder not so good

→ Start scripts in external shell




Run configuration per file

Select a run configuration:

/home/ck/.config/spyder-py3/temp.py

Console

○ Execute in current console

○ Execute in a dedicated console

● Execute in an external system terminal

General settings

□ Clear all variables before execution

□ Directly enter debugging when errors appear

□ Command line options:

Working Directory settings

● The directory of the file being executed

○ The current working directory

○ The following directory:

External system terminal

☑ Interact with the Python console after execution

□ Command line options:

□ Always show this dialog on a first file run

Excercise: Embedded IPython

Listing: ipython1.py

```
import math

# import embedded shell
from ipydex import IPS

a = 10
b = 20.5
c = a + b + 3**2
d = math.sqrt(c)

# run embedded shell
IPS()
# try: math.sqrt?, math.s<TAB>, history (up, down), %magic
# exit with CTRL-D
```

Summary + Outlook

What we talked about

- Course organization
- Usage of Jupyter-Notebook, Spyder
- Python basics (see `01_overview_Python_types_and_syntax.pdf`)
- Exercise

How to continue?

- Review of the material before the next lesson