

# Toolbox Workshop

Nützliche Programme für Physikstudenten

Igor Babuschkin   Kevin Dungs   Christian Gerhorst

Peter Lorenz   Ismo Toijala

**PeP et al. e.V.**

September 2012



**PEP ET AL. E.V.**  
PHYSIKSTUDIERENDE UND  
EHMALIGE PHYSIKSTUDIERENDE  
DER TU DORTMUND

[www.pep-dortmund.org](http://www.pep-dortmund.org)

*Der Verein versteht sich als Einrichtung für Absolventen, Studierende, Mitarbeiter sowie für Freunde und Förderer der Fakultät Physik der TU Dortmund. Gegründet auf Initiative einiger Absolventen ist es seine Aufgabe, ein Netzwerk zwischen den Absolventen und der Fakultät aufzubauen.*

# Motivation

- Arbeitserleichterung
- "Gute" Tools
  - Kein Excel!
- Teamarbeit++
- *best practices*

Unix Shell

git

Python

## Unix Shell

Dateisystem

Befehle

Nützliche Shell Features

- / trennt Teile eines Pfads
- Das gesamte Dateisystem bildet *einen* Baum
- Es gibt immer ein aktuelles Verzeichnis
- Pfade können absolut oder relativ angegeben werden
- drei spezielle Verzeichnisse:
  - . das aktuelle Verzeichnis
  - .. das Oberverzeichnis des aktuellen Verzeichnisses
  - ~ das Heimverzeichnis
- Dateien, die mit . anfangen sind versteckt

## man, pwd, cd

- `man topic` für manual: zeigt die Hilfe für ein Programm
- `pwd` für print working directory: zeigt das aktuelle Verzeichnis
- `cd directory` für change directory: wechselt in das angegebene Verzeichnis

- ls [*directory*] für list: zeigt den Inhalt eines Verzeichnisses an
- ls -l zeigt mehr Informationen über Dateien und Verzeichnisse
- ls -a zeigt auch versteckte Dateien



# Tastaturkürzel

`Ctrl-C` beendet das laufende Programm

`Ctrl-D` EOF (end of file) eingeben, kann Programme beenden

`Ctrl-L` leert den Bildschirm

# Globber (\*)

- \* wird ersetzt durch alle passenden Dateien



## git

Warum?

Befehle

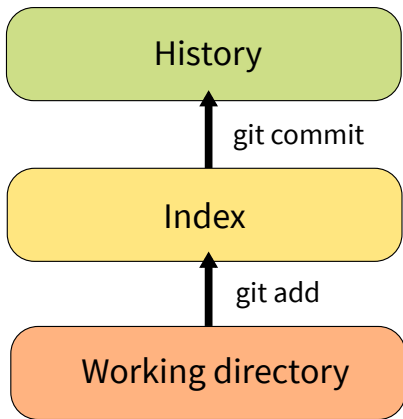
Hoster

## Warum Versionskontrolle?

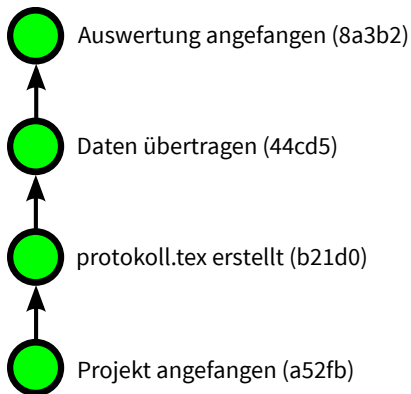
- Backup
- vereinfachte Kollaboration
- Protokollierung

## Warum Git?

- *distributed* Version Control System
- sehr schnell
- setzt sich momentan durch



# Übersicht



# Repository erstellen

- `git init` Erzeugt ein leeres Repository im jetzigen Ordner
- `git clone [url]` Kopiert ein Repository aus dem Internet

# Informationen abrufen

`git status`    Zeigt an, welche Dateien geändert wurden und welche bereits im Index sind

`git log`        Zeigt alle gespeicherten Commits an



## Dateien zum Index hinzufügen

- `git add` Fügt eine Datei oder einen Ordner (mit Inhalt) zum Index hinzu
- `git rm [-r]` Löscht eine Datei aus dem Ordner und schreibt die Löschung in den Index
- `git mv` Genauso, aber die Datei wird verschoben

# Commits erstellen

`git commit` Speichert die Änderungen im Index als  
Commit ab

# Änderungen runter-/hochladen

- `git pull` Neue Commits runterladen  
Falls man noch neue lokale Commits hat  
führt git einen *merge* durch
- `git push` Neue Commits hochladen  
Geht nur, wenn keine neuen Commits im  
zentralen Repository sind  
Wenn ja, erst einmal `git pull` verwenden

# Manuell mergen

`git mergetool` Startet ein Programm, mit dem man manuell mergen kann, falls die Automatik nicht funktioniert

-  **GitHub** (<https://github.com/>)  
kostenlose öffentliche Repositories, gute UI
-  **Bitbucket** (<https://bitbucket.org>)  
kostenlose private Repositories



[www.python.org](http://www.python.org)

*Python is a programming language that lets you work more quickly and integrate your systems more effectively. You can learn to use Python and see almost immediate gains in productivity and lower maintenance costs.*

# Python ist...

- ... eine Programmiersprache.
- ... einfach!
- ... sehr mächtig.
- ... universell einsetzbar.



<http://xn--sptzleimitsoss-cfb.de/wp-content/uploads/2012/07/eierlegendewollmilchsau.jpg>

# Ein kleines Beispiel

## C++

```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    cout << "Hello, World!" << endl;
    return 0;
}
```

## Python

```
print("Hello, World!")
```



## Python

IPython

Sprache

Bibliotheken

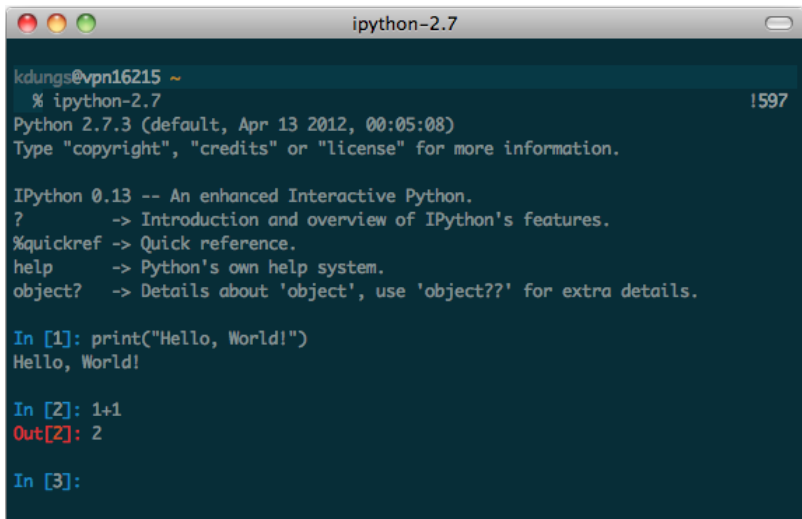
PyLab

NumPy

SciPy

matplotlib

# IPython



```
ipython-2.7
kdungs@vpn16215 ~
% ipython-2.7                                     !597
Python 2.7.3 (default, Apr 13 2012, 00:05:08)
Type "copyright", "credits" or "license" for more information.

IPython 0.13 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.

In [1]: print("Hello, World!")
Hello, World!

In [2]: 1+1
Out[2]: 2

In [3]:
```

## Blöcke

- Durch Einrückung!
- 4 Leerzeichen (/1 Tab)

## Semikolons

- Gibt es prinzipiell
- Sind am Zeilenende aber nicht notwendig

# Variablen

- Dynamische Typisierung
- Keine explizite Deklaration

## Beispiel

```
In [1]: a = 1
```

```
In [2]: b = 2
```

```
In [3]: name = "Käähbiin"
```

```
In [4]: a, b, name
```

```
Out [4]: (1, 2, 'Käähbiin')
```

# Datenstrukturen

- bool (True, False)
- int, float, long, complex
- string ('foo', "bar")
- Iteratoren, Generatoren, Sequenzen, ...

## Praktische Typen

() Tupel

[] Liste

{ } Dictionary

## Zum Beispiel

```
In [9]: cities = ['Dortmund', 'Hamburg', 'Berlin']
```

```
In [10]: cities[0]
```

```
Out[10]: 'Dortmund'
```

# Mehr Beispiele

```
In [1]: teams = {  
    ...:     'BVB': "BV Borussia Dortmund 09",  
    ...:     'S04': "FC Schalke 04",  
    ...:     'FCB': "FC Bayern München"  
    ...: }
```

```
In [2]: teams['BVB']
```

```
Out[2]: 'BV Borussia Dortmund 09'
```

# Operatoren

## Arithmetische Operatoren

`+`, `-`, `*`, `/`, `%`, `**`, `//`

## Zuweisungsoperatoren

`=`, `+=`, `-=`, `*=`, `/=`, `%=`, `**=`, `//=`

## Vergleichsoperatoren

`==`, `!=`, `<`, `<=`, `>`, `>=`



# Operatoren

## Logische Operatoren

`and`, `or`, `not`

## Identitätsoperatoren

`is`, `is not`

## Operatoren für Sequenzen

`in`, `not in`

if

```
if condition:
```

```
    # do something
```

```
elif other_condition:
```

```
    # or do something else
```

```
else:
```

```
    # or something else
```

case

gibt es *nicht!*

while

```
while condition:
```

```
    # do something
```

for

- for ... in
- agiert immer auf *Sequenzen!*

# for - Beispiele

## Listen

```
for city in cities:  
    print(city)
```

Dortmund  
Hamburg  
Berlin

## Range

```
for i in range(0, 10):  
    print(i)
```

0  
1  
2  
3

...

## Aufruf

- `print(something)`
- `funktionsname(*args, **kwargs)`

## Definition

```
def funktionsname(param1, param2=defaultwert, ...):  
    # do something
```

- `import modulname`
- `import modulname as alias`
- `from modulname import teil`
- `from modulname import *`

## Zum Beispiel

```
import numpy as np  
a = np.array([1, 2, 3])
```

# Objektorientierung

- gibt es auch!
- Klassen, Objekte, Methoden
- Juhu!

## Und ohne IPython?

- Dateiendung `.py`
- Aufruf per `python3 dateiname.py`
- `print()`-Funktion



# Beispiele und Aufgaben herunterladen

Lade das Repository

`https://github.com/ibab/toolbox`

per `git clone` herunter.

- bündelt NumPy, SciPy und Matplotlib
- starten mit `ipython3 --pylab`



- $n$ -dimensionale Arrays
- Funktionen, die auf denen arbeiten
- Operatoren wirken elementweise
- wird meist mit `np` abgekürzt

Konstanten:

- `pi`
- `e`

# Arrays erstellen

- `array`: konvertiert irgendwas (Liste, Tupel, ...) zu einem Array
- `linspace(start, end, number)`: Array aus `number` Zahlen zwischen `start` und `end` in gleichem Abstand
- `arange(start, end, step)`: Array aus Zahlen zwischen `start` und `end` mit dem Abstand `step`
- `zeros(shape)`: Array aus Nullen der Größe `shape`
- `ones(shape)`: Array aus Einsen der Größe `shape`

# Elementweise Funktionen

Beispiele:

- `sqrt`
- `exp, log`
- `sin`
- `deg2rad, rad2deg`

# Reduzierende Funktionen

Beispiele:

- `sum`
- `mean`
- `max, min`
- `ediff1d`

- `loadtxt(file [, unpack=True])`: Lädt eine Datei in ein Array. `unpack=True` transponiert das Array
- `savetxt(file, array)`: Speichert ein Array in eine Datei



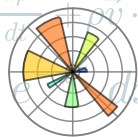


# Nützliche Funktionen

- `optimize.curve_fit`: fittet nichtlineare Funktionen
- `stats.sem`: gibt den Fehler des Mittelwerts
- `constants.C2K`: konvertiert Celsius in Kelvin
- `constants.K2C`: konvertiert Kelvin in Celsius

# Konstanten

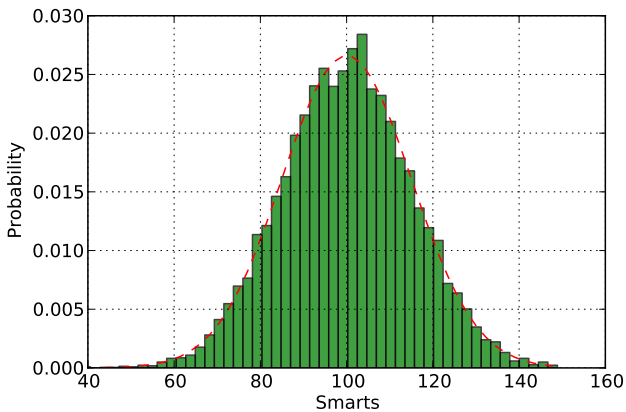
- `constants.physical_constants`: enthält diverse physikalische Konstanten, ihre Fehler und Einheiten (aus CODATA)



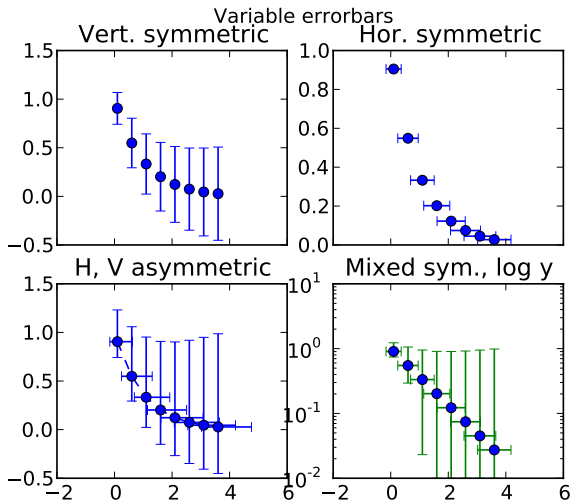
# matplotlib

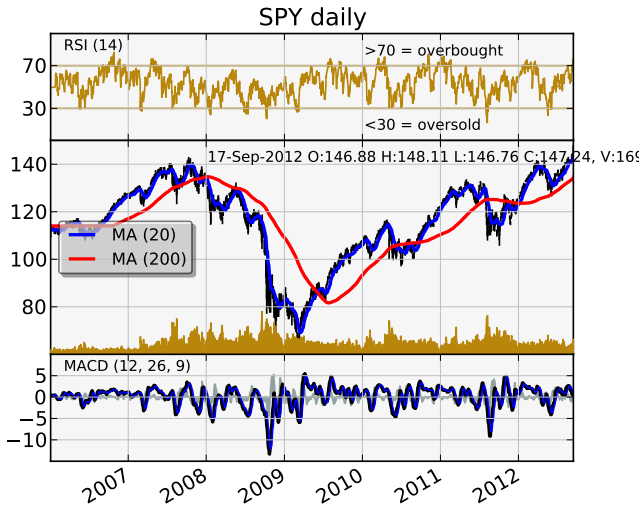
- prozedurales Interface (pyplot, in PyLab) — einfacher
- objektorientiertes Interface (Beschreibung im Skript)  
— flexibler, schöner für größere Skripte oder Programme

# Beispiele

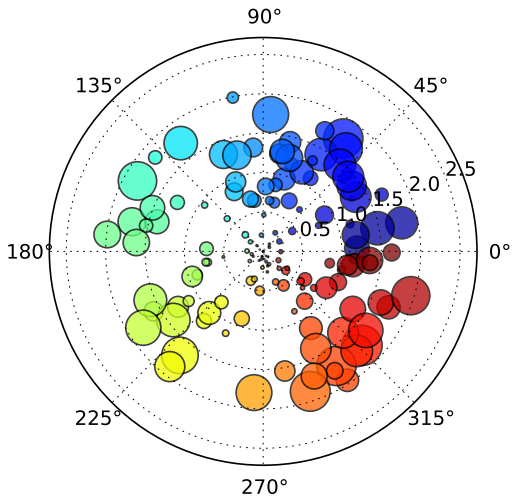


# Beispiele



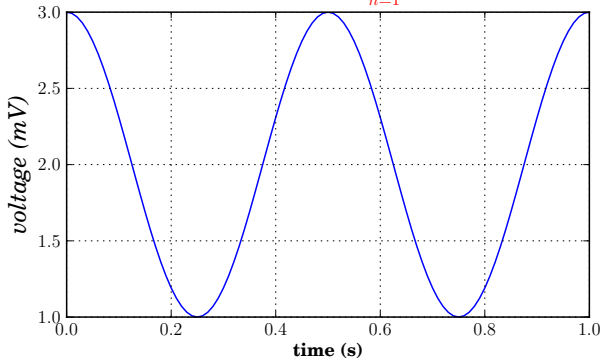


# Beispiele



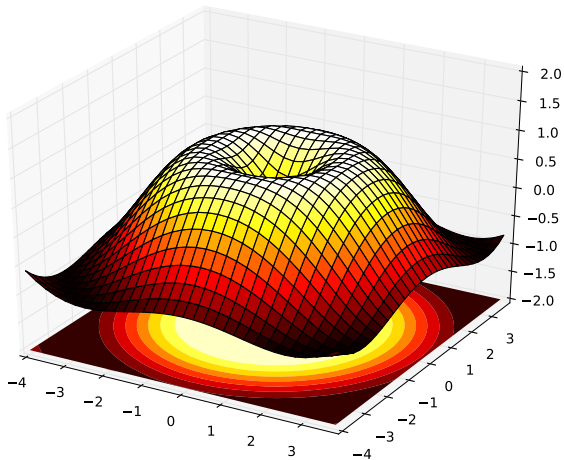
# Beispiele

$\TeX$  is Number  $\sum_{n=1}^{\infty} \frac{-e^{i\pi}}{2^n}!$





# Beispiele



# Plotten mit IPython

Einfach `ipython3 --pylab` ausführen und z.B. Folgendes eintippen:

```
In [1]: x = linspace(0, 1, 100)
```

```
In [2]: plot(x, x**2, 'b-')
```

An dem Plot kann interaktiv weiter gearbeitet werden.

# Plots in .py-Dateien erstellen

Erst einmal Bibliotheken importieren.

Plot erscheint, wenn man `show()` aufruft.

```
from numpy import linspace, pi, sin
import matplotlib.pyplot as plt
x = linspace(0, 2 * pi, 1000)
plt.plot(x, sin(x), 'r--')
plt.show()
# oder: plt.savefig('plot.pdf')
```

# Verschiedene Arten von Plots

- `plot`
- `errorbar` - Plot mit Fehlerbalken
- `semilogy`, `semilogx` - logarithmische Skalierung
- `hist` - Histogramme
- `polar` - polare Plots

# Nützliche Funktionen

- `title('...')`
- `xlabel('...'), ylabel('...')`
- `grid()`
- `xlim(a, b), ylim(a, b)`
- `legend()` (beim Plot `label='...'` einstellen)
- `clf()`

# Matplotlib einstellen

2 Möglichkeiten:

- direkt in der Code-Datei
- Datei `matplotlibrc` im selben Ordner

⇒ siehe Dokumentation

# Vielen Dank für's Zuhören!

