Toolbox Workshop

PeP et al. Toolbox Workshop

PeP et al. e.V.

Physikstudierende und
ehemalige Physikstudierende
der TU Dortmund

2023

Auf das Praktikum vorbereiten

Daten: Abspeichern Auswerten Visualisieren

Zusammenarbeiten Protokoll verfassen

Technische Fähigkeiten, die man in der Wissenschaft braucht

Konkrete Probleme durch Programmieren lösen

Wiederholte Abläufe automatisieren

Versionskontrolle: Wieso? und Wie?

Kooperation mit Anderen an gemeinsamen Projekten

Kommandozeile

Von Anfang an: Best Practices

Spart Zeit und Nerven

Verwenden von Dokumentation

Erleichtert Zusammenarbeit mit Anderen

Was sind die Standardwerkzeuge?

Offene, Reproduzierbare Wissenschaft

- → Große Teile der Wissenschaft stecken in einer „Replikationskrise“
- → Gründe sind vielfältig, aber hauptsächlich
  - → Publikations-Bias (positive Ergebnisse werden veröffentlicht, Null-Resultate nicht)
  - → Zugrunde liegende Daten nicht verfügbar
  - → Genutzte Software nicht verfügbar
  - → Mangelhafte statistische Analyse
  - → Veröffentlichungen mit der vollen Information hinter „Paywalls“
  - → ...
- → Das muss besser gehen!

Open Science Grundprinzipien

1.	Öffentliche Daten	Open Data
2.	Öffentlicher Quellcode	Open Source
3.	Öffentliche Methodik	Open Methodology
4.	Öffentlicher „Peer Review“ Prozess	Open Peer Review
5.	Öffentlicher Zugang zu Veröffentlichungen	Open Access
6.	Öffentliche Bildungsangebote	Open Educational Resources

Reproduzierbarkeit

- Öffentliche Software
- Öffentliche Daten
- Automatisierung so weit wie es irgendwie geht
- Versionskontrolle

FAIR Prinzipien für Daten

F Findable

A Accessible

I Interoperable

R Reusable

Kurz: Wissenschaftliche Daten müssen in wohldefinierten, offenen Datenformaten mit Metadaten, die den Inhalt des Datensatzes beschreiben, abgespeichert werden.

Mehr infos hier: `https://www.go-fair.org/fair-principles/`

Der Toolbox Workshop

- Einführung in einen zusammenpassenden Satz von Werkzeugen um reproduzierbare, offene Wissenschaft zu ermöglichen
- Das Praktikum soll im Kleinen die Grundlagen des wissenschaftlichen Arbeitens vermitteln
  ⇒ Als Chance sehen, die hier vorgestellten Konzepte zu üben
- Spätestens essentiell bei Bachelor- und Masterarbeit
- Nützlich weit darüber hinaus

Der Toolbox Workshop

- Wir zeigen *eine mögliche* Kombination von Tools
- Für alle Bereiche gibt es andere Möglichkeiten mit Vor- und Nachteilen
- Die hier gezeigten Tools sind aber sehr weit verbreitet, auch außerhalb der Wissenschaft

Toolbox Workshop

Python

- Höhere, stark und dynamisch typisierte, universelle, interpretierte Programmiersprache
- Beliebteste Programmiersprache in der Wissenschaft seit einigen Jahren
- Großes Ökosystem an wissenschaftlichen Bibliotheken
- Aber auch: Webentwicklung, Automatisierung, Systemprogrammierung, Spiele, etc.
- Einsteigerfreundlich, aber auch komplex

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

Weitere wichtige Programmiersprachen: C++, C, Java, Fortran, Rust, Julia, R

Python Installation

- Es gibt (leider) viele Varianten Python und Bibliotheken für Python zu installieren
- Weit verbreitet in der Wissenschaft: die Paketmanager „conda“ und „mamba“
- mamba ist eine wesentlich effizientere Implementierung des conda Tools, wird hoffentlich bald per default in conda integriert.[1]

[1] Siehe `https://www.anaconda.com/blog/conda-is-fast-now`

Isolierte Umgebungen

- Wichtig: Isolierung von Software für verschiedene Projekte um Konflikte zu vermeiden und feste Versionen zu installieren (Reproduzierbarkeit!)
- conda / mamba ermöglichen es, isolierte, benannte Umgebungen zu erzeugen mit eigenen Versionen von Python und Bibliotheken
- Umgebungen müssen „aktiviert“ werden um die Software darin zu nutzen:

```
$ which python
/usr/bin/python
$ mamba activate toolbox
$ which python
/home/maxnoe/.local/conda/envs/toolbox/bin/python
$ mamba deactivate
```

- Umgebungen können in YAML Textdateien definiert werden
- Empfehlung: für jedes Projekt (zum Beispiel das Anfängerpraktikum) eine eigene Umgebung aufsetzen und `environment.yaml` Datei im Projekt speichern

Automatisierung

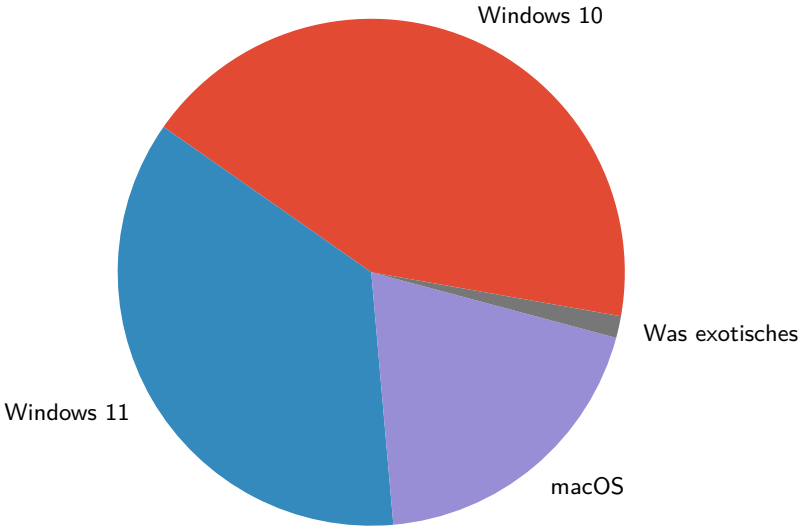
- Reproduzierbarkeit erfordert Dokumentation und größtmögliche Automatisierung, da bei jedem Schritt Fehler gemacht werden können
- Wir zeigen die Software „GNU Make“
- Komplexe Abfolge von einzelnen Schritten wird in einer Textdatei beschrieben
- Führt alle notwendigen Schritte mit einem einzelnen Aufruf des `make` Befehls aus
- Wichtiger Zusammenhang mit der Kommandozeile:
  - Makefiles verknüpfen Inputs mit Outputs über Kommandozeilen-Befehle
  - Grafische Benutzeroberflächen sind zwar „schön“ und „intuitiv“ aber extrem schwierig zu automatisieren / reproduzierbar zu machen

Versionskontrolle und Kooperation mit Git

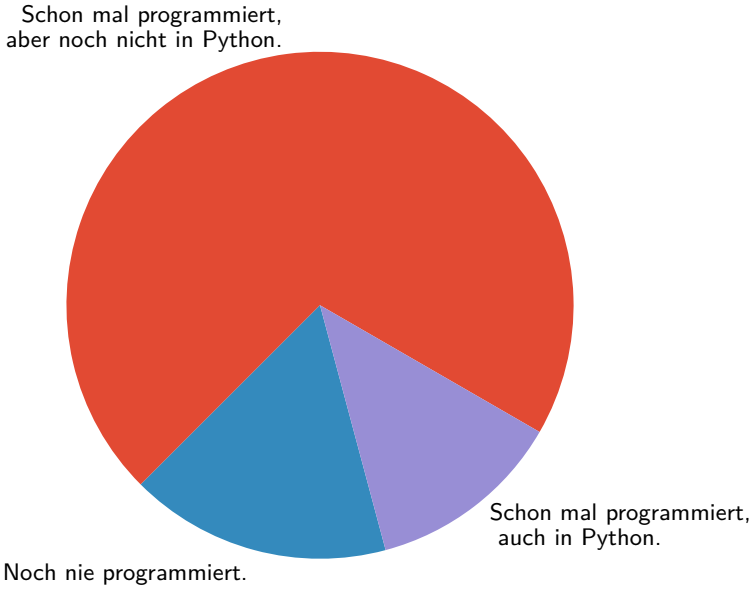
- → Versionskontrolle mit Tools wie Git ist der zentrale Punkt von offener, reproduzierbarer Wissenschaft
- → Genutzt in der Entwicklung quasi aller Open Source Software
- → Ermöglicht Kooperation von mehreren Personen am gleichen Projekt
- → Verküpft Änderungen mit Begründungen / Erklärungen
- → Ermöglicht Rückkehr zu älteren Versionen
- → Synchronisierung zwischen mehreren Rechnern
- → Backup!
- → Hosting-Provider wie GitHub und GitLab ermöglichen öffentliche und private Projekte

Ergebnisse der Umfrage

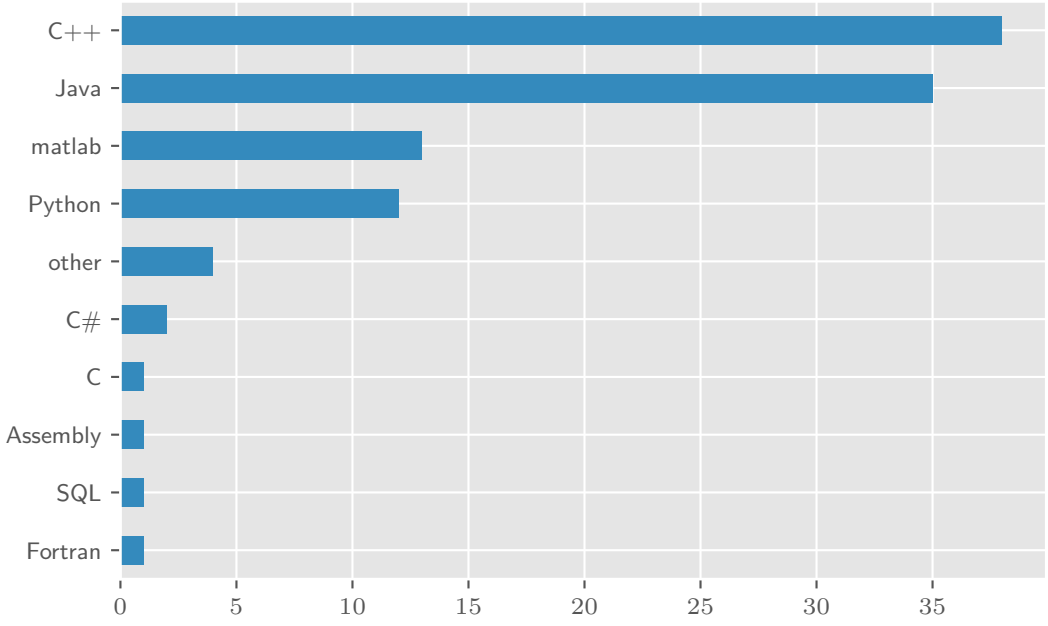
Betriebssystem



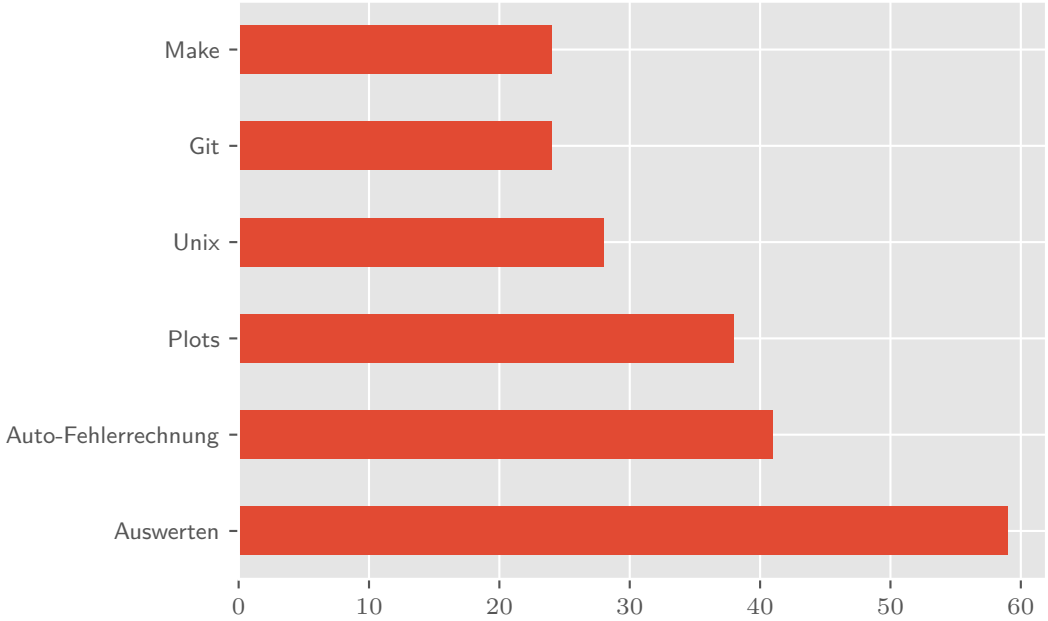
Programmierkenntnisse



Programmiersprachen



Interessen


Make
Git
Unix
Plots
Auto-Fehlerrechnung
Auswerten
0
10
20
30
40
50
60

Ablauf

Montag Programmieren mit Python

Dienstag Erstellen von Plots / Auswerten

- → NumPy
- → matplotlib

Mittwoch Auswerten / Fehlerrechnung

- → scipy
- → uncertainties

Donnerstag Kommandozeile und Automatisierung

- → Unix
- → make

Freitag Versionskontrolle

- → git
- → Abschließende Übungen

Nächste Woche Verfassen wissenschaftlicher Texte mit LaTeX

- → Fließtext & Mathematik
- → Referenzen & Literaturverzeichnis
- → Integration mit dem Inhalt der ersten Woche
- → Vollständige Übungen und Vorlage fürs Praktikum

And Now for Something Completely Different

Betriebssysteme

Betriebssysteme

Proprietäres Betriebssystem von Microsoft

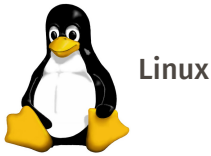
Läuft auf den meisten Geräten

NT Kernel

Proprietäres Betriebssystem von Apple

Läuft nur auf Apple Geräten

Unix / BSD



Open Source Betriebssystem *Kernel*

Läuft auf den meisten Geräten

Unix

Viele verschiedene *Distributionen*

Viele kommerzielle Software unterstützt nur Windows und/oder macOS.

Man kann mehrere Betriebssysteme auf dem gleichen Rechner installieren. Nativ (Dual-Boot) oder in „Virtuellen Maschinen“.

Windows 10 & 11 bringen das *Windows Subsystem for Linux* mit, eine integrierte Linux VM.

Linux-Distributionen

Eine Linux-Distribution kombiniert den Linux-Kernel mit weiterer Software. Hauptsächlich:

- → Desktop-Umgebung(en)
- → Paket-Manager und zugehörige Server mit Software

Es gibt viele „Familien“ von Linux-Distributionen, die sich die gleichen oder ähnliche Tools teilen:

Für den Einstieg empfehlen wir die aktuellen Versionen von Ubuntu oder Fedora.

Alle zwei Jahre erscheint eine Ubuntu-Version mit 5 Jahren Support (LTS), aktuell: 22.04.

`https://distrowatch.com/`

Texteditoren

Was haben die mit diesem Kurs zu tun?

Texteditoren

- Viele Dateien, denen man in der Wissenschaft begegnet, enthalten (plain) text
  - Paper/Arbeiten mit LaTeX
  - Programm-Code
  - Config-Files
  - Notizen
  - Daten (csv, json, yaml, ...)
  - Emails
- Es lohnt sich also, einen guten Texteditor zu wählen und den Umgang damit zu erlernen!
- Das spart auf lange Sicht Zeit und macht die Arbeit angenehmer
- Zwei Varianten: Terminal / GUI

Textdateien und Unicode

Was ist eigentlich eine Textdatei?

- → In einer Datei stehen immer Binärdaten in Bytes, 1 Byte = 8 Bit, 0-255
- → Es gibt (gab) viele Varianten, Text in Binärdaten umzuwandeln (Encoding)
- → Heute sollte immer Unicode enkodiert als `utf-8` verwendet werden
- → Es gibt viele standardisierte Dateiformate, die auf Textdateien basieren
  json, yaml, toml, ecsv, ...
- → Und weniger standardisierte aber trotzdem verbreitete Formate:
  csv, fixed width table, ...

Unicode
- → Sammlung von Schriftzeichen, Buchstaben, Akzente, Emojis, ...
- → Aus allen Sprachen.
- → Ordnet Zeichen „Codepoints“ zu
- → Beispiele: `LATIN SMALL LETTER A`: 97, `PILE OF POO`: 128169

UTF-8 Encoding um Unicode-Text in Bytes zu speichern

Zeilenende

Windows und Unix-Systeme verwenden unterschiedliche Konventionen für ein Zeilenende.

Unix `\n` / `LF` (Linefeed)

Windows `\r\n` / `CR LF` (Carriage Return + Linefeed).

VS Code/VS Codium erkennt auf allen Betriebssystemen, welche Konvention in der aktuellen Datei genutzt wird und behält sie bei.

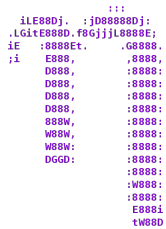
Empfehlung: immer Unix-Konvention nutzen

Was muss ein Editor können?

In absteigender Wichtigkeit

- → Zeilennummern
- → Syntax-Highlighting
- → Simple Autovervollständigung
- → Plugins / Anpassbarkeit
- → Linting (Warnhinweise für falschen Code)
- → Komplexe Autovervollständigung (Snippets, Library-Funktionen)

Nano



- → Einfacher Texteditor fürs Terminal
- → Auf fast jedem Unix-System vorhanden
- → Wenige Features, nicht erweiterbar

Vim

- → Moden-basiert
- → Erweiterbar
- → Auf fast jedem Unix-System default
- → Harter Einstieg

Visual Studio Code

- → GUI Editor von Microsoft
- → Leichter zu bedienen
- → Batteries included
- → Viele nützliche Plugins

Obligatory XKCD → http://xkcd.com