

Toolbox Workshop

PeP et al. Toolbox Workshop



PeP et al. e.V.

Physikstudierende und
ehemalige Physikstudierende
der TU Dortmund

2024

Auf das Praktikum vorbereiten

Daten: Abspeichern Auswerten Visualisieren

Zusammenarbeiten Protokoll verfassen Automatisieren

Technische Fähigkeiten, die man im Praktikum/in der Wissenschaft braucht

Konkrete Probleme durch Programmieren lösen

Wiederholte Abläufe automatisieren

Versionskontrolle: Wieso? und Wie?

Kooperation mit Anderen an gemeinsamen Projekten

Von Anfang an: Ein Werkzeugkasten

Spart Zeit und Nerven

Verwenden von Dokumentation

Erleichtert Zusammenarbeit mit Anderen

Was sind die Standardwerkzeuge?

Der Toolbox Workshop

- Einführung in einen zusammenpassenden Satz von Werkzeugen, um gute Wissenschaft zu ermöglichen (offen, reproduzierbar)
- 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

Daten

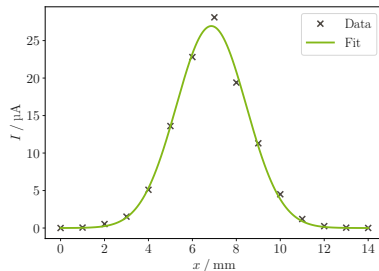
x / mm	$I / \mu\text{A}$
0	0,000
1	0,060
2	0,530
3	1,520
4	5,100
\vdots	\vdots

Daten

x / mm	$I / \mu\text{A}$
0	0,000
1	0,060
2	0,530
3	1,520
4	5,100
\vdots	\vdots



Plots und Ergebnisse



$$I_0 = (26,9 \pm 0,5) \mu\text{A}$$

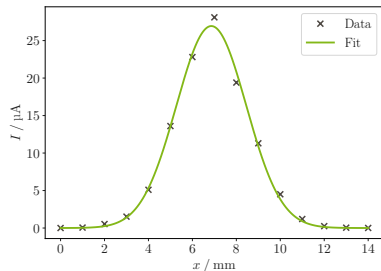
Daten

x / mm	$I / \mu\text{A}$
0	0,000
1	0,060
2	0,530
3	1,520
4	5,100
\vdots	\vdots

 python



Plots und Ergebnisse

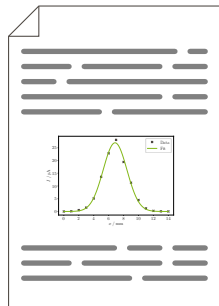


$$I_0 = (26,9 \pm 0,5) \mu\text{A}$$

\LaTeX



Protokoll (PDF)



Montag Programmieren mit Python

Dienstag Datenhandhabung / Erstellen von Plots

- NumPy
- matplotlib

Mittwoch Datenauswertung / Fehlerrechnung

- scipy
- uncertainties

Donnerstag Kommandozeile und Versionskontrolle

- Unix
- git

Freitag Einstieg in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$

Nächste Woche Verfassen wissenschaftlicher Texte mit $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$

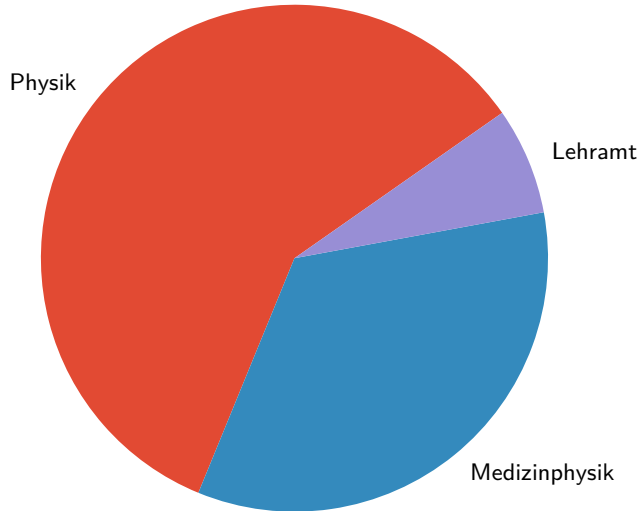
- Fließtext & Mathematik
- Referenzen & Literaturverzeichnis

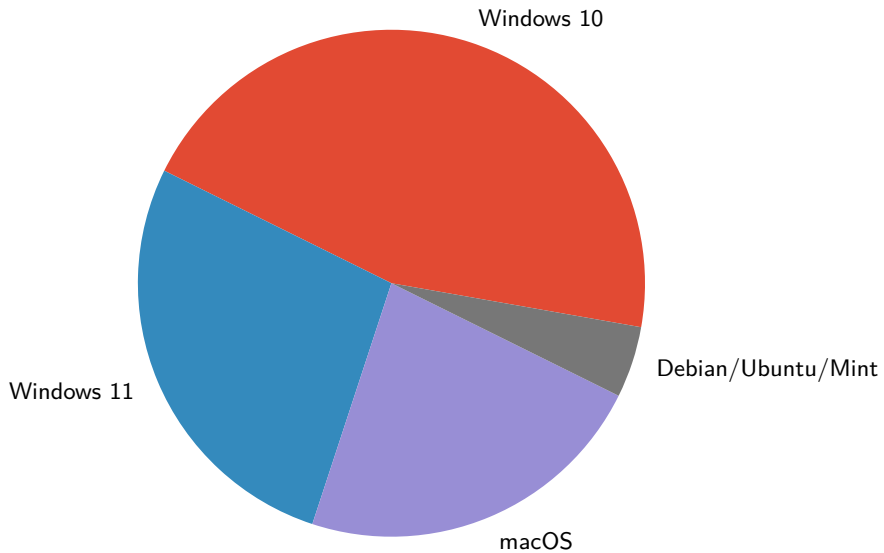
Automatisierung mit make

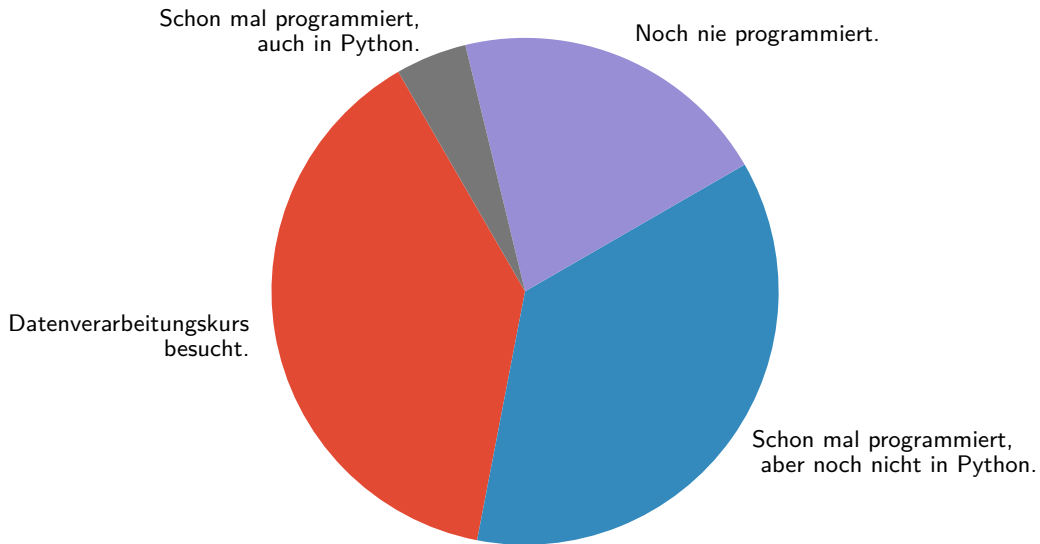
Kombination aller gezeigten Tools

Protokollvorlage und abschließende Übungen

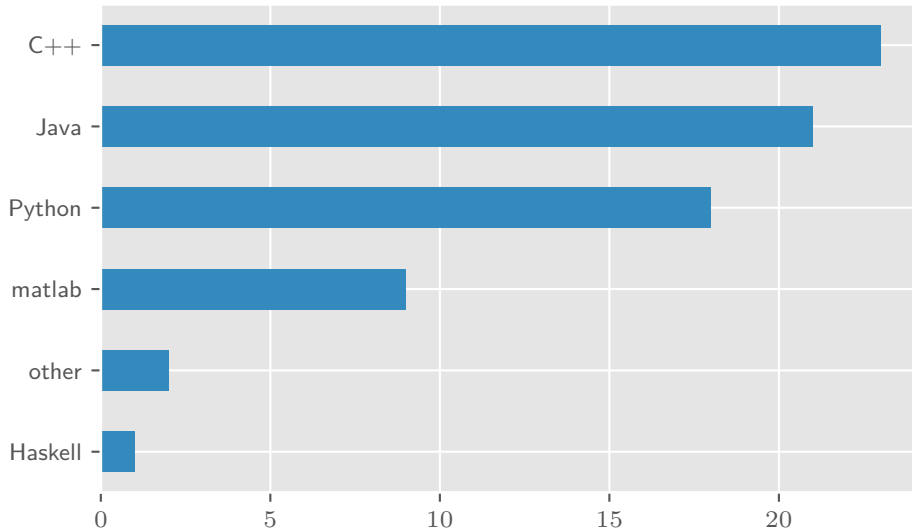
Ergebnisse der Umfrage



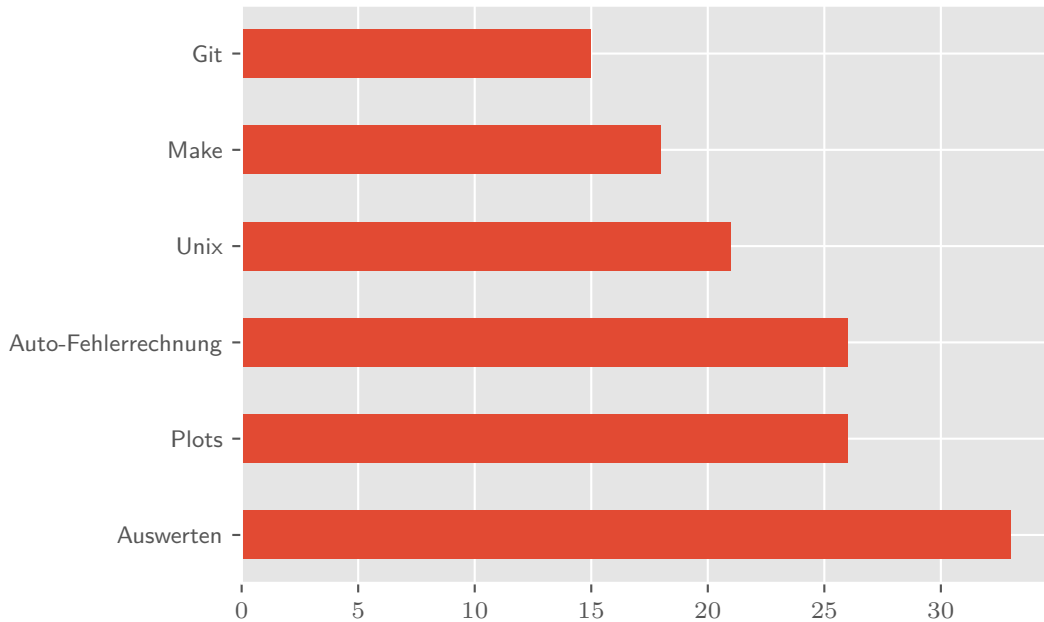




Programmiersprachen



Interessen



Betriebssysteme



Proprietäres Betriebssystem von
Microsoft

Läuft auf den meisten Geräten
NT Kernel



Proprietäres Betriebssystem von
Apple

Läuft nur auf Apple Geräten
Unix / BSD



Linux

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: 24.04.

<https://distrowatch.com/>

Warum wir (in der Mehrheit) Linux benutzen:

- Freiheit in der Auswahl der Software, nicht proprietär
- Flexibilität und Personalisierung im Aussehen und in der Handhabung
- Mehr Kontrolle über die eigenen Daten, **kein** Account oder Internetzugang bei der Installation notwendig
- **Kein** überwachendes oder „helfendes“ KI System im OS
- Gewohnheit aus der (längeren) Zeit an der Uni (teilweise Interaktion mit Servern notwendig; diese sind meist ein Linux-artiges System)

Warum bei uns *alles anders* aussieht:

- Selbstkonfigurierte, nicht unbedingt standardmäßige Desktop-Umgebung
- (teilweise) Kein Standard Ubuntu
- **VIEL** investierte Zeit in das eigene System

Dennoch:

Alle Funktionen, die wir euch zeigen, funktionieren auch bei euch.

Texteditoren

Was haben die mit diesem Kurs zu tun?

- Viele Dateien, denen man in der Wissenschaft begegnet, enthalten (plain) text
 - Paper/Arbeiten mit $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$
 - Programm-Code
 - 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

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

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)

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)

Windows Notepad:

```
def fit(x: np.ndarray, i0: float, mu: float, sigma: float) -> np.ndarray:
    """Gaussian Fit function.

    Parameters
    -----
    x : array_like
        x data.
    i0 : float
        Amplitude.
    mu : float
        Mean.
    sigma : float
        Standard deviation
    """
    return i0 * np.exp(-2 * (x - mu) ** 2 / sigma**2)

params, _ = curve_fit(fit, x, y)
x_lin = np.linspace(0, 14, 1000)

fig, ax = plt.subplots(figsize=(7, 5), layout="constrained")
ax.plot(x, y, "x", ms=8, mew=2, color="#4d4742", label="Data")
ax.plot(x_lin, fit(x_lin, *params), color="#838818", label="Fit")

ax.set(
    xlabel=r"$x \text{ \mathbin{/} } \text{ \unit{ \milli\metre} }$",
    ylabel=r"$I \text{ \mathbin{/} } \text{ \unit{ \micro\ampere} }$",
)
ax.legend()

fig.savefig("./build/example_plot.pdf")
```

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)

Windows Notepad:

```
def fit(x: np.ndarray, i0: float, mu: float, sigma: float) -> np.ndarray:
    """Gaussian Fit function.

    Parameters
    -----
    x : array_like
        x data.
    i0 : float
        Amplitude.
    mu : float
        Mean.
    sigma : float
        Standard deviation
    """
    return i0 * np.exp(-2 * (x - mu) ** 2 / sigma**2)

params, _ = curve_fit(fit, x, y)
x_lin = np.linspace(0, 14, 1000)

fig, ax = plt.subplots(figsize=(7, 5), layout="constrained")
ax.plot(x, y, "x", ms=8, mew=2, color="#4d4742", label="Data")
ax.plot(x_lin, fit(x_lin, *params), color="#838818", label="Fit")

ax.set(
    xlabel=r"$x \text{ \textbackslash mathbin{ } \text{ \unit{ \milli metre} } }$",
    ylabel=r"$I \text{ \textbackslash mathbin{ } \text{ \unit{ \micro ampere} } }$",
)
ax.legend()

fig.savefig("../build/example_plot.pdf")
```

VS Codium:

```
21 (function) def linspace(
22 x, y = np.g start: ArrayLikeNumber,
23 stop: ArrayLikeNumber,
24 num: SupportsIndex = ...,
25 def fit(x): endpoint: bool = ...,
26 """Gaus retstep: Literal[False] = ...,
27 Paramet dtype: DtypeLike = ...,
28 -----) axis: SupportsIndex = ...,
29 ) -> ndarray
30 x : arr
31 x 0 Return evenly spaced numbers over a specified interval.
32 i0 : fl Returns num evenly spaced samples, calculated over the interval [ start, stop ].
33 Amp The endpoint of the interval can optionally be excluded.
34 mu : fl
35 Pico versionchanged
36 sigma : 1.16.0 Non-scalar start and stop are now supported.
37 Sta new
38 versionchanged
39 return 1.20.0 Values are rounded towards -inf instead of 0 when an integer dtype is specified. The old behavior can still be
40 obtained with np.linspace(start, stop, num, dtype=int).
41
42 params, _ = Parameters
43 x_lin = np.linspace(0, 14, 1000)
44
45
46 fig, ax = plt.subplots(figsize=(7, 5), layout="constrained")
47
48 ax.plot(x, y, "x", ms=8, mew=2, color="#4d4742", label="Data")
49 ax.plot(x_lin, fit(x_lin, *params), color="#838818", label="Fit")
50
51 ax.set(
52     xlabel=r"$x \text{ \textbackslash mathbin{ } \text{ \unit{ \milli metre} } }$",
53     ylabel=r"$I \text{ \textbackslash mathbin{ } \text{ \unit{ \micro ampere} } }$",
54 )
55 ax.legend()
56
57 fig.savefig("../build/example_plot.pdf")
```

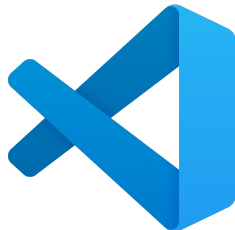
Nano

```
      :::  
iLE88dj. :jd88888dj:  
.LGitE888D.f8GjjjL888E;  
iE :8888Et. .G8888.  
;i  E888,      ,8888,  
    D888,      :8888:  
    D888,      :8888:  
    D888,      :8888:  
    D888,      :8888:  
    888W,      :8888:  
    W88W,      :8888:  
    W88W,      :8888:  
    DGGD:      :8888:  
                :8888:  
                :W888:  
                :8888:  
                E888i  
                tW88D
```

(Neo)Vim



Visual Studio Code

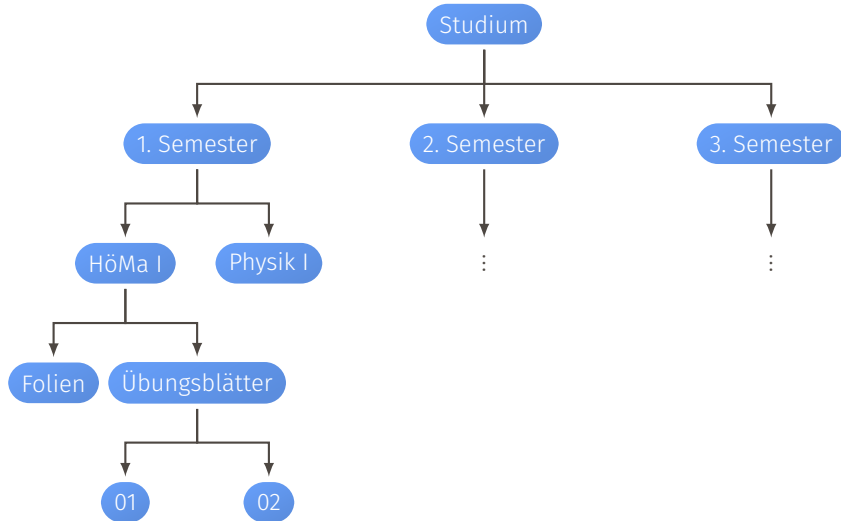


- Einfacher Texteditor fürs Terminal
- Auf fast jedem Unix-System vorhanden
- Wenige Features, nicht erweiterbar

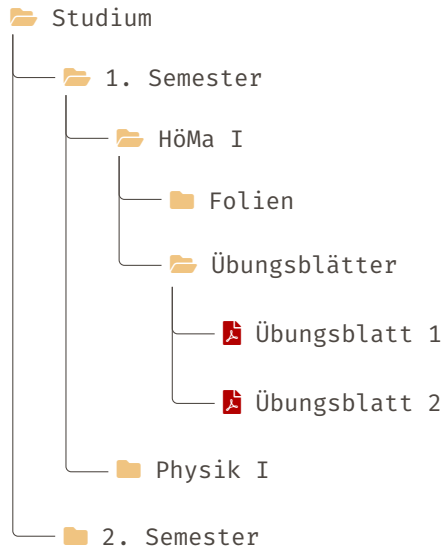
- Modi-basiert
- Erweiterbar
- Auf fast jedem Unix-System default
- Harter Einstieg

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

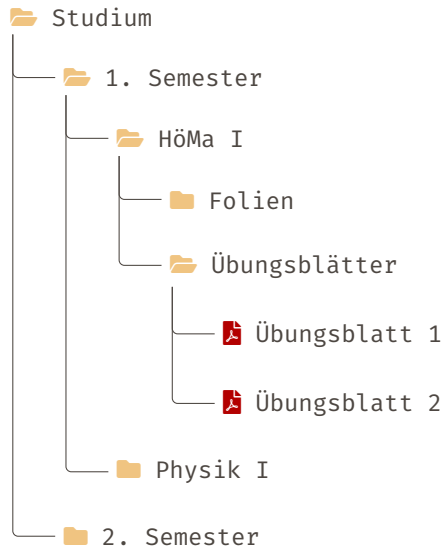
Ordnerstruktur



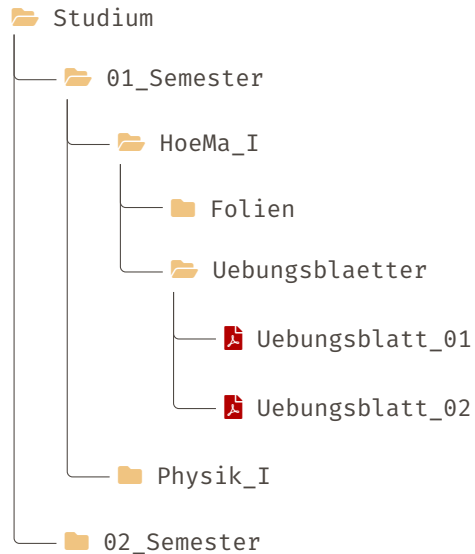
Beispiel einer typischen Ordnerstruktur



Beispiel einer typischen Ordnerstruktur



Anpassung der Namen:



<code>ls</code>	„list“: Zeigt den Inhalt eines Verzeichnisses
<code>cd <dirname></code>	„change directory“ wechselt in das Verzeichnis <dirname>
<code>cd ..</code>	wechselt in das Oberverzeichnis
<code>cd -</code>	wechselt in das vorherige Verzeichnis
<code>mamba activate toolbox</code>	lädt die installierten python Pakete
<code>codium .</code>	öffnet VSCodium im aktuellen Verzeichnis (Linux/Mac)
<code>code .</code>	öffnet VSCode im aktuellen Verzeichnis (Windows)
<code>python vorlage.py</code>	führt „vorlage.py“ mit python aus
<code>lualatex main.tex</code>	führt „main.tex“ mit \LaTeX aus