

# Werkzeuge für das wissenschaftliche Arbeiten

## *Python for Machine Learning and Data Science*

Magnus Bender  
[bender@ifis.uni-luebeck.de](mailto:bender@ifis.uni-luebeck.de)  
Wintersemester 2022/23

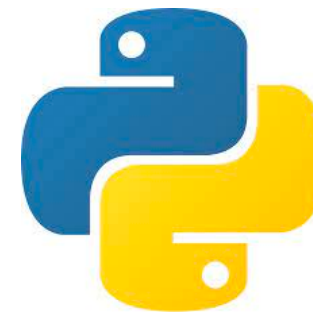
# Inhaltsübersicht

## 1. Programmiersprache Python

a) *Einführung, Erste Schritte*

b) *Grundlagen*

c) *Fortgeschritten*



## 2. Auszeichnungssprachen

a) *LaTeX, Markdown*

L<sup>A</sup>T<sub>E</sub>X



## 3. Benutzeroberflächen und Entwicklungsumgebungen

**a) Jupyter Notebooks lokal und in der Cloud (Google Colab)**

## 4. Versionsverwaltung

a) Git, GitHub



## 5. Wissenschaftliches Rechnen

a) NumPy, SciPy



## 6. Datenverarbeitung und -visualisierung

a) Pandas, matplotlib, NLTK

## 7. Machine Learning (scikit-learn)

a) Grundlegende Ansätze (Datensätze, Auswertung)

b) Einfache Verfahren (Clustering, ...)



## 8. DeepLearning

a) TensorFlow, PyTorch, HuggingFace Transformers



# Themen

- I. Projektaufgabe 1
  1. Lösungsvorschlag
- II. Benutzeroberflächen und Entwicklungsumgebungen
  1. Jupyter Notebooks
  2. Grundlagen (Bash-)Terminal
- III. Reguläre Ausdrücke



*Heute*

# Projektaufgabe 1

## „Textbasierter Taschenrechner“

- Vorstellung möglicher Lösungen  
*(werden nicht in Moodle hochgeladen!)*



# II.

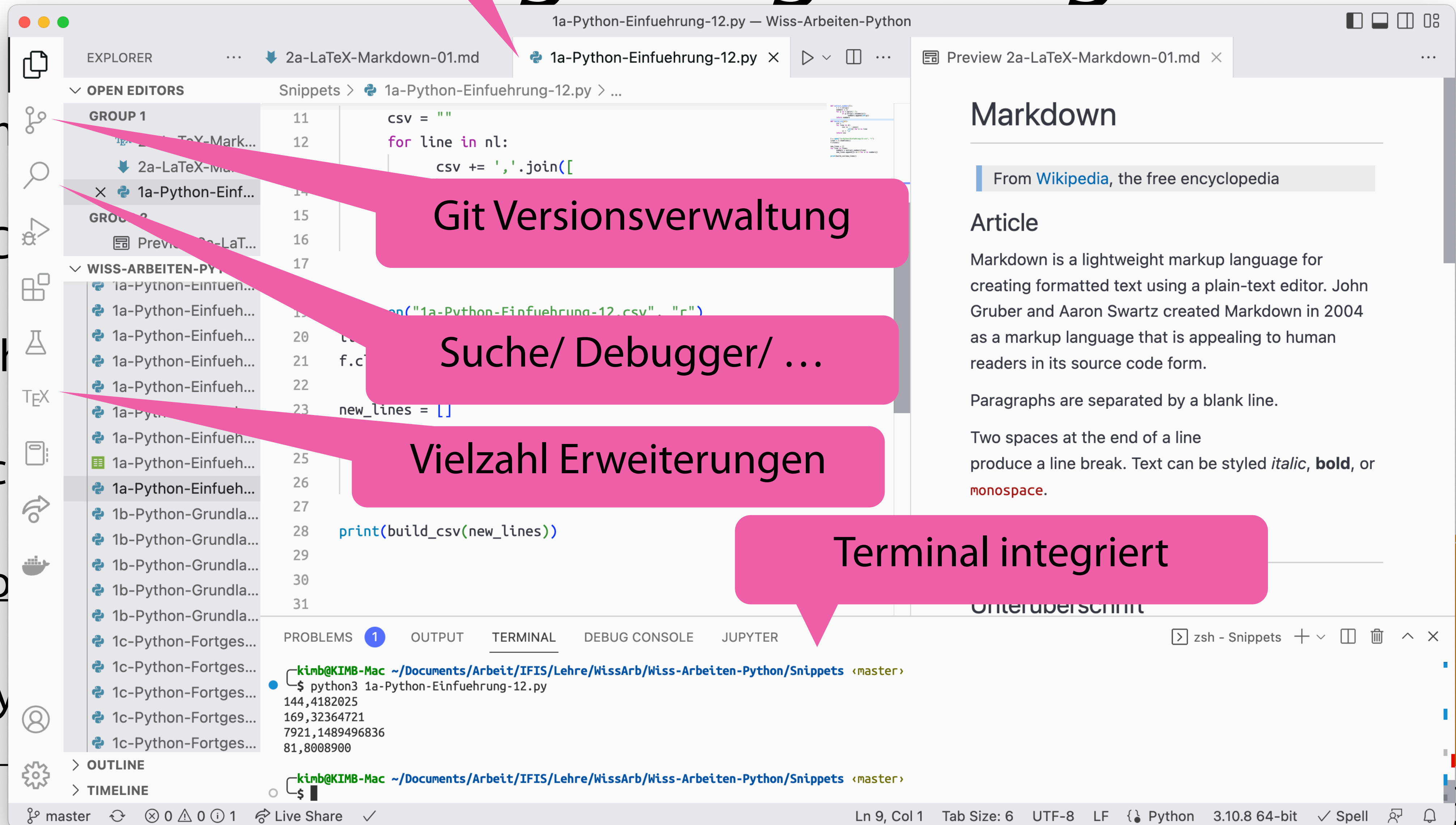
# Benutzeroberflächen und Entwicklungsumgebungen

## *1. Jupyter Notebooks – lokal und in der Cloud*

Editortabs mit verschiedenen Inhalten

# ENTWICKLUNGsumgebungen

- Vielzahl
- VS C
- PyCh
- Entwic
- VSCO
- Jupy



Git Versionsverwaltung

Suche/ Debugger/ ...

Vielzahl Erweiterungen

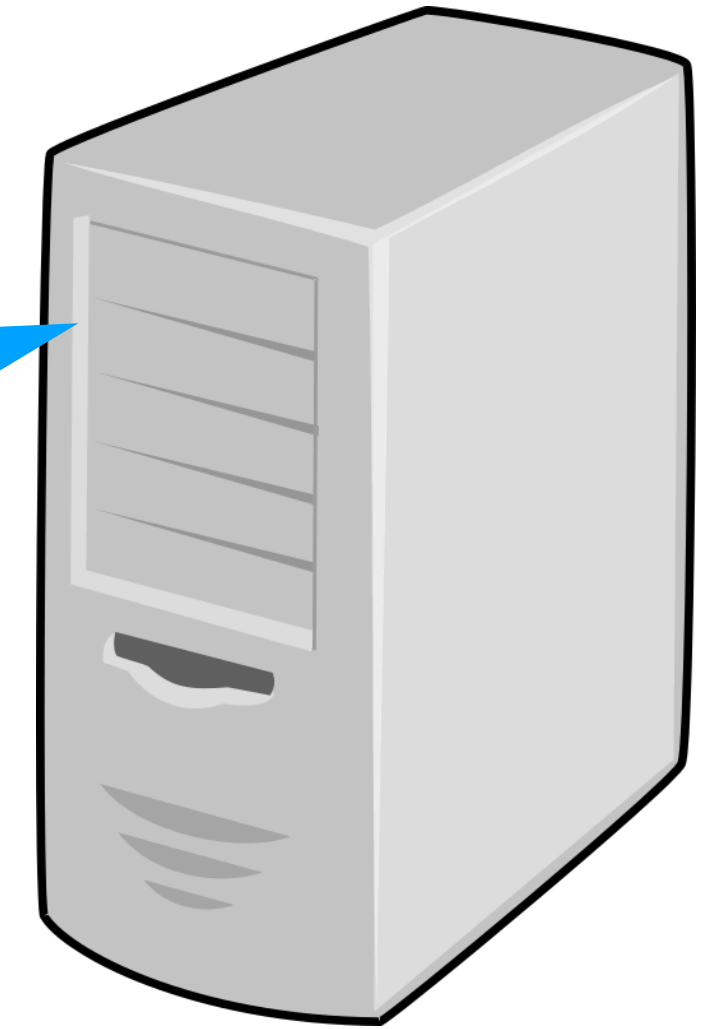
Terminal integriert



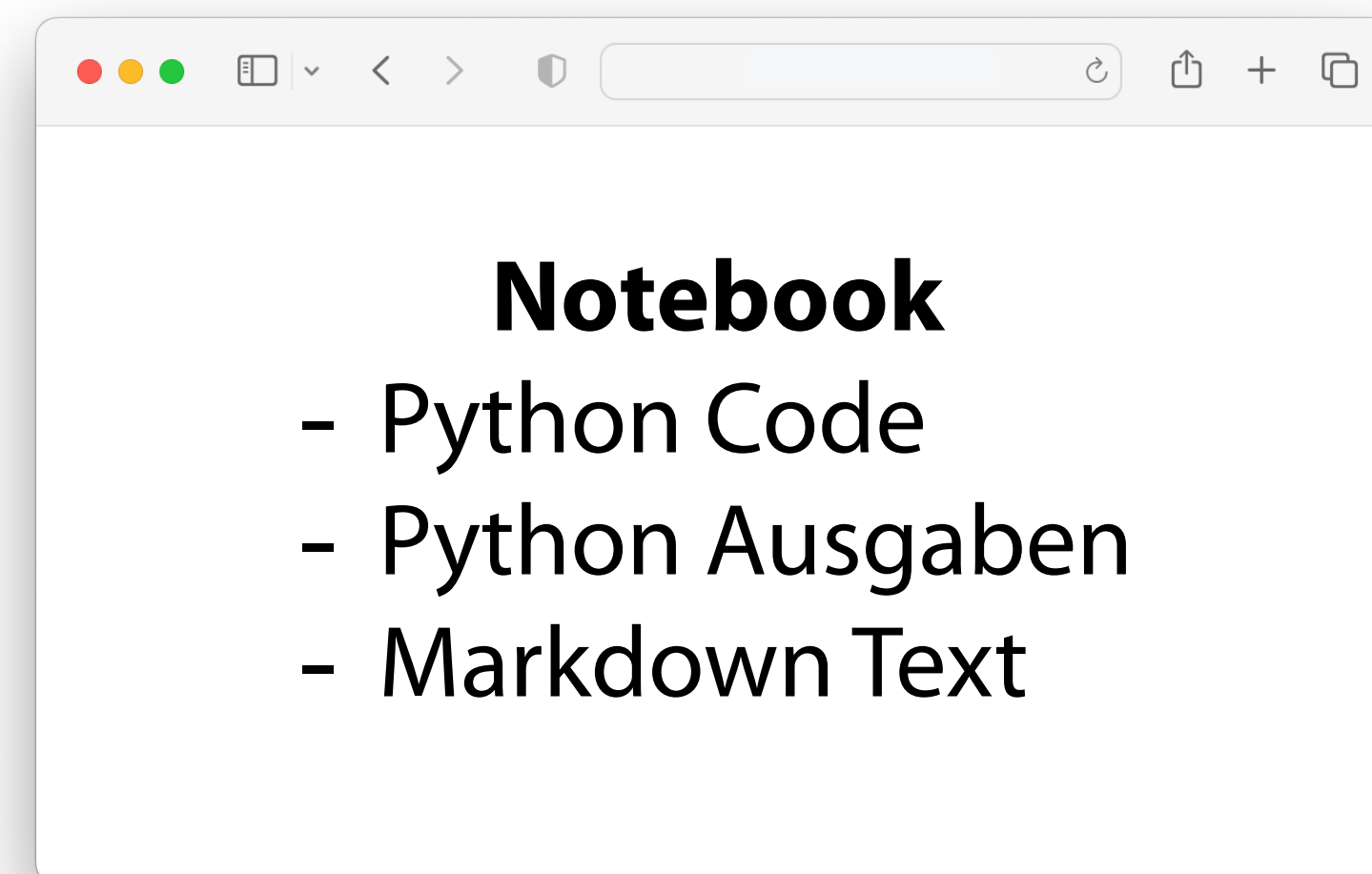
# Jupyter Notebooks

- Darstellung im Browser (aber u.a. auch in VS Code)
- Installation lokal oder Nutzung in der Cloud
- Kombination von Programm und Text (Markdown)

Kernel (Python o.a.)



HTTP Verbindung  
(lokal oder über das Internet)



# Notebook Beispiel

Ein Notebook besteht aus mehreren Zellen, jede Zelle kann einen anderen Inhaltstypen haben:

- Python Code
- Markdown Text

Und es werden z.B. auch Formeln unterstützt!

$$\sum_{i=1}^n i = \frac{n(n+1)}{2} = \frac{n^2 + n}{2}$$

```
In [1]: import sys
        sys.version
```

```
Out[1]: '3.10.6 (main, Nov 2 2022, 18:53:38) [GCC 11.3.0]'
```

Einen Codeblock kann man mittels `Shift+Enter` ausführen, die Ausgabe erscheint dann direkt darunter (auch ohne `print()`).  
Übrigens, die Eingabe von Markdown beendet man auch mit `Shift+Enter`.

```
In [2]: def summe(n):
        s = 0
        for i in range(n+1):
            s += i
        return s
```

Eine Funktion kann einfach definiert werden und ist dann im *laufenden* Kernel verfügbar, also in allen weiteren Codeblöcken definiert.

```
In [3]: "Korrekt" if summe(10) == (10**2+10)/2 else "Fehlerhaft"
```

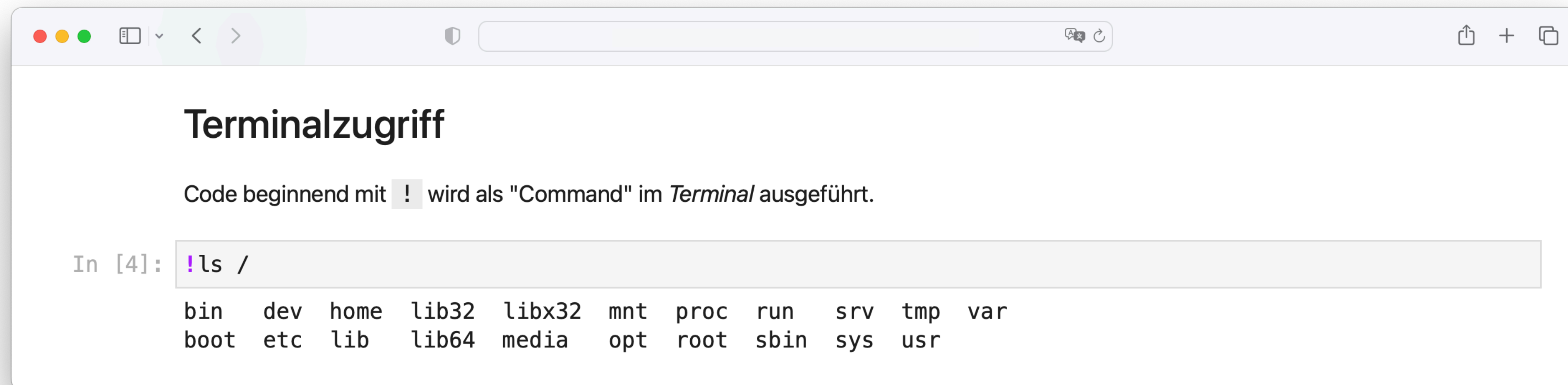
```
Out[3]: 'Korrekt'
```



# Jupyter in der Cloud

- Beispiel: Google Colab(oratory)
- Kostenfreie Nutzung
- Zugriff auf GPUs

Praktisch bei der Entwicklung von Code und Erklärung bei gleichzeitiger Anzeige von Ausgaben.



The screenshot shows a terminal window titled "Terminalzugriff". Below the title, it says "Code beginnend mit ! wird als "Command" im Terminal ausgeführt." Below that, the command `!ls /` is entered in a text box. The output of the command is displayed below the text box:

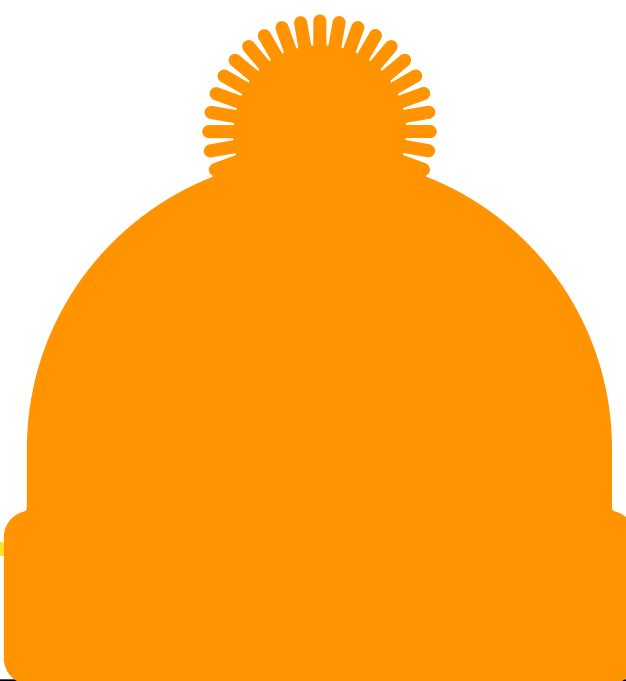
```
bin  dev  home  lib32  libx32  mnt  proc  run  srv  tmp  var
boot  etc  lib  lib64  media  opt  root  sbin  sys  usr
```

# II.

# Benutzeroberflächen und Entwicklungsumgebungen

## *1. Die Shell und das Terminal*

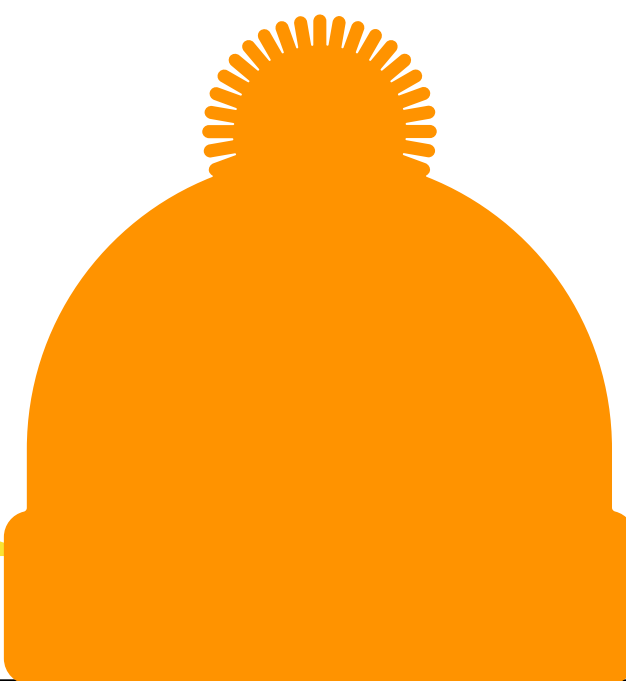
Wir hatten zuvor immer Beispiele mit Befehlen für das Terminal – aber haben das Terminal nie betrachtet.



# Unixoides Betriebssystem

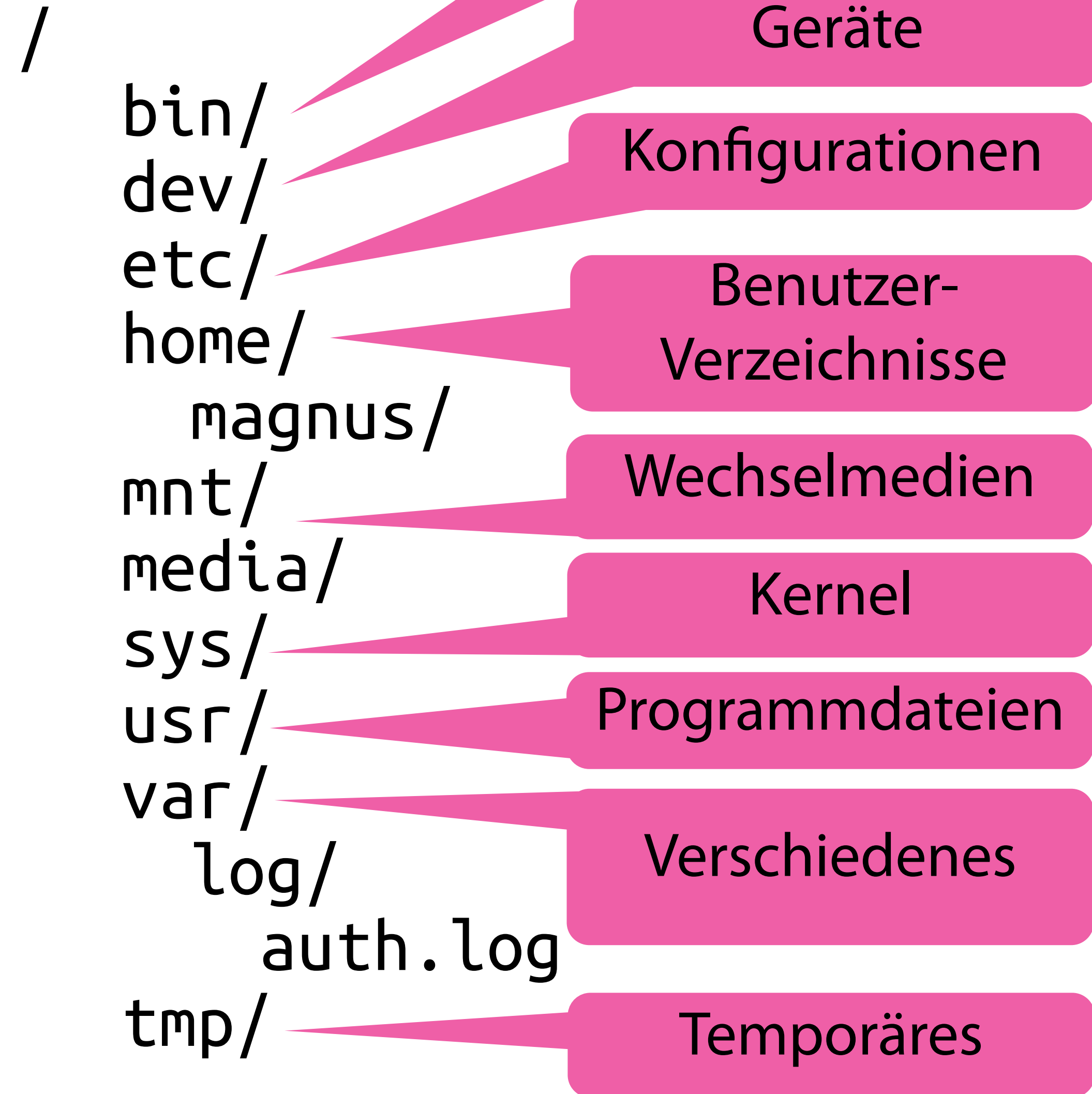
- Grundzüge
  - „Alles ist eine Datei“
  - „Unix-Shell“
  - Benutzerverwaltung (Admin = root)
  - Grafikserver X11
  - ...
- Beispiele
  - Linux, macOS, BSD
  - Auf Windows Linux über die WSL (Windows Subsystem for Linux) möglich

Wir beziehen uns im folgenden auf eine „Unix-Shell“



# Pfade

- „/“ Wurzelpfad
- „.“ referenziert das aktuelle Verzeichnis
- „..“ referenziert das übergeordnete Verzeichnis
- „~“ referenziert das Home-Verzeichnis des aktuellen Users
- Versteckte (Datei-)Namen beginnen mit „.“



# Dateisystem

```
user@user-virtual-machine: ~  
user@user-virtual-machine:~$ ls -la /  
total 2097240  
drwxr-xr-x 20 root root 4096 Feb 8 2021 .  
drwxr-xr-x 20 root root 4096 Feb 8 2021 ..  
drwxrwxrwx 7 root root 4096 Feb 8 2021 bin -> usr/bin  
drwxr-xr-x 4 root root 4096 Nov 18 23:45 boot  
drwxrwxr-x 2 root root 4096 Feb 8 2021 cdrom  
drwxr-xr-x 18 root root 4180 Nov 18 23:45 dev  
drwxr-xr-x 139 root root 12288 Nov 18 23:44 etc  
drwxr-xr-x 4 root root 4096 Feb 8 2021 home  
drwxrwxrwx 7 root root 4096 Feb 8 2021 lib -> usr/lib  
drwxrwxrwx 9 root root 4096 Feb 8 2021 lib32 -> usr/lib32  
drwxrwxrwx 9 root root 4096 Feb 8 2021 lib64 -> usr/lib64  
drwxrwxrwx 10 root root 4096 Feb 8 2021 libx32 -> usr/libx32  
drwxr-xr-x 16384 root root 16384 Feb 8 2021 lost+found  
drwxr-xr-x 2 root root 4096 Feb 10 2021 media  
drwxr-xr-x 2 root root 4096 Apr 23 2020 mnt  
drwxr-xr-x 2 root root 4096 Mär 19 2021 opt  
-xr-xr-x 434 root root 0 Nov 18 23:45 proc  
drwxr-xr-x 4 root root 4096 Apr 6 2021 root  
drwxr-xr-x 3 root root 980 Nov 18 23:45 run  
drwxrwxrwx 8 root root 4096 Feb 8 2021 sbin -> usr/sbin  
drwxr-xr-x 1 root root 4096 Nov 24 2021 snap  
drwxr-xr-x 2 root root 4096 Apr 23 2020 srv  
w----- 1 root root 147483648 Feb 8 2021 swapfile  
-xr-xr-x 1 root root 0 Nov 18 23:45 sys  
drwxrwxrwt 1 root root 4096 Nov 18 23:49 tmp  
drwxr-xr-x 14 root root 4096 Apr 23 2020 usr  
drwxr-xr-x 14 root root 4096 Apr 23 2020 var
```

→ Dateityp

```
kimb — kimb@KIMB-Mac — ~ — -zsh — 105x34  
Last login: Fri Nov 18 23:30:58 on ttys002  
kimb@KIMB-Mac ~  
$ ls -la /  
total 9  
drwxr-xr-x 20 root wheel 640 Oct 28 10:43 .  
drwxr-xr-x 20 root wheel 640 Oct 28 10:43 ..  
lrwxr-xr-x 1 root admin 36 Oct 28 10:43 .VolumeIcon.icns -> System/Volumes/Data/.Vol  
----- 1 root admin 0 Oct 28 10:43 .file  
drwxr-xr-x 2 root wheel 64 Oct 28 10:43 .vol  
drwxrwxr-x 55 root admin 1760 Nov 18 22:17 Applications  
drwxr-xr-x 66 root wheel 2112 Nov 10 17:38 Library  
drwxr-xr-x@ 10 root wheel 320 Oct 28 10:43 System  
drwxr-xr-x 5 root admin 160 Nov 10 17:36 Users  
drwxr-xr-x 4 root wheel 128 Nov 18 19:14 Volumes  
drwxr-xr-x@ 39 root wheel 1248 Oct 28 10:43 bin  
drwxr-xr-x 2 root wheel 64 Dec 13 2019 cores  
dr-xr-xr-x 3 root wheel 4514 Nov 18 19:13 dev  
lrwxr-xr-x@ 1 root wheel 11 Oct 28 10:43 etc -> private/etc  
lrwxr-xr-x 1 root wheel 25 Nov 18 19:13 home -> /System/Volumes/Data/home  
drwxr-xr-x 5 root nixbld 160 May 7 2022 nix  
drwxr-xr-x 3 root wheel 96 Aug 8 2020 opt  
drwxr-xr-x 6 root wheel 192 Nov 18 19:13 private  
drwxr-xr-x@ 64 root wheel 2048 Oct 28 10:43 sbin  
lrwxr-xr-x@ 1 root wheel 11 Oct 28 10:43 tmp -> private/tmp  
drwxr-xr-x@ 11 root wheel 352 Oct 28 10:43 usr  
lrwxr-xr-x@ 1 root wheel 11 Oct 28 10:43 var -> private/var
```



# „Command“

- Programmaufruf über das Terminal

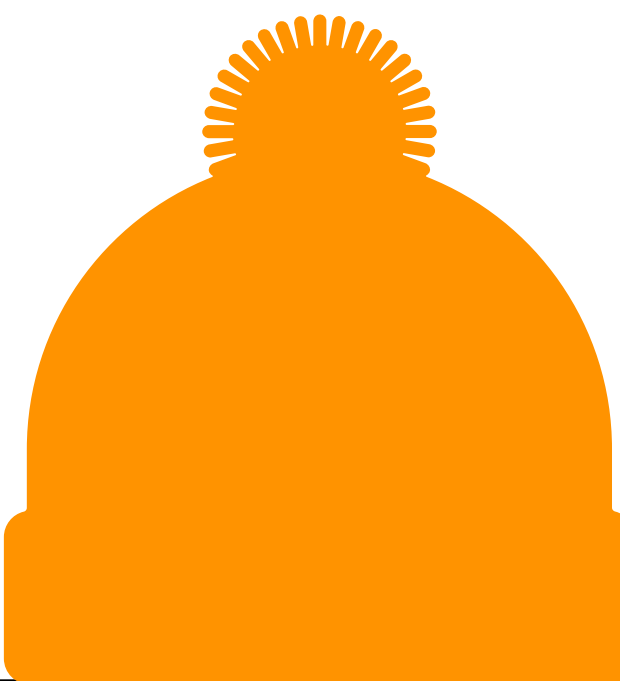
Leerzeichen als Trenner

```
programm [[-p <wert>] --parameter <wert>] argument
```

- Programmname
- Optionale Parameter
- Argumente

Häufig haben Parameter eine kurze (-) und eine Lange (- -) Bezeichnung.

Bei kurzen Parametern kann man meist statt „programm -a -b“ auch „programm -ab“ ausführen.

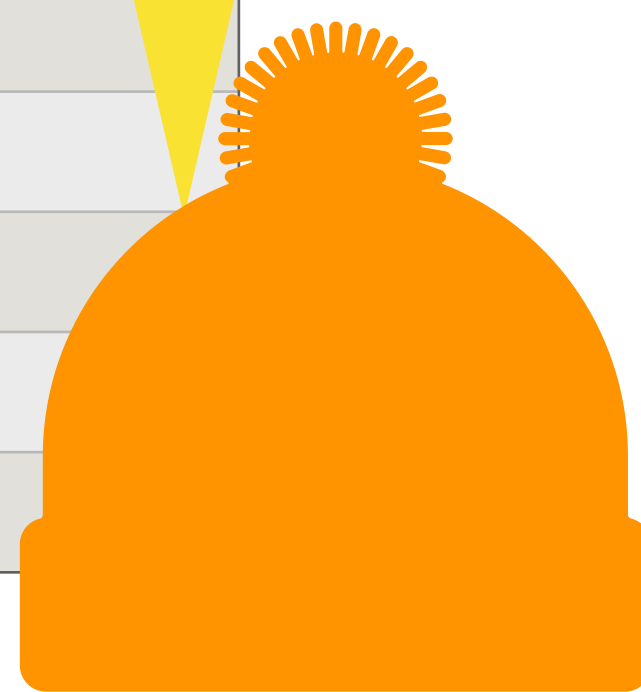


# Nützliche Befehle I

|                                 |        |                              |  |
|---------------------------------|--------|------------------------------|--|
| Navigation<br>im<br>Dateisystem | cd     | cd ..                        | In einen anderen Ordner wechseln           |
|                                 | ls     | ls .                         | Inhalte eines Ordners auflisten            |
|                                 | pwd    | pwd                          | Aktuellen Pfad anzeigen                    |
|                                 | mc     | mc .                         | Midnight Commander (GUI-artiger Explorer)  |
|                                 | find   | find . -name "*.txt" -type f | Suchen und finden von Dateien              |
|                                 | grep   | grep "hi" ./file.txt         | Suchen und finden in Dateien               |
| Dateien                         | touch  | touch ./new.txt              | Erstellen einer leeren Datei               |
|                                 | mcedit | mcedit ./file.txt            | Bearbeiten einer Datei (GUI-artig, von mc) |
|                                 | nano   | nano ./file.txt              | Bearbeiten einer Datei (Quasi-Standard)    |
|                                 | vim    | vim ./file.txt               | Bearbeiten einer Datei                     |
|                                 | rm     | rm ./file.txt                | Löschen einer Datei (oder auch Ordnern!)   |

Mit „rm -rf /“ würde man das ganze Dateisystem rekursiv löschen (sofern man die Berechtigung hat).

„How to exit Vim?“



# Nützliche Befehle II

|                   |                      |   |                        |
|-------------------|----------------------|---|------------------------|
| Systemüberwachung | <code>top</code>     | Zeigt aktuell laufende Prozesse   |                        |
|                   | <code>htop</code>    | Task-Manager (GUI-artig)  |                        |
|                   | <code>free</code>    | Zeigt Speicherbelegung an   |                        |
|                   | <code>df -h</code>   | Zeigt Festplatten und Belegung an   |                        |
| Hilfe             | <code>man</code>     | <code>man ls</code>   | Öffnet Hilfeseite      |
|                   | <code>whereis</code> | <code>whereis python</code>   | Zeigt Pfad zu Programm |
|                   | <code>whoami</code>  | Bestimmt aktuellen Nutzernamen  |                        |
| Netzwerk          | <code>wget</code>    | <code>wget <a href="http://example.com/file.zip">http://example.com/file.zip</a></code> |                        |
|                   | <code>curl</code>    | <code>curl -I <a href="http://example.com">http://example.com</a></code>                |                        |



# Das Terminal

Nutzer      Host      Aktueller Pfad

```
user@a98a624173f9:~/pyweb/scripts$
```

Vervollständigung mittels Tabulator-  
taste (u.a. Programm-, Dateinamen).

Mit den Pfeiltasten ↑ ↓ kann  
man durch seine letzten  
Commands navigieren – mit  
Ctrl+R darin suchen.

Abbruch mittels Ctrl+C

- Führt eine Shell aus
- Ausführungsumgebung für Commands
- Viele Designs, ähnlicher Aufbau

# Symbole > & | >> &

Leitet Ausgabe auf Datei um (> überschreibt und >> fügt an).

- Um- und weiterleiten

```
echo "Hallo Welt" > ./file.txt  
echo "Tschuess" >> ./file.txt
```

file.txt

```
Hallo Welt  
Tschuess
```

```
echo "1,2,3" | grep "2"  
cat ./file.txt | python ./process.py
```

Pipe, hier: STDOUT des ersten Programms in STDIN des zweiten

- Prozesse

```
sleep 1 && echo "Hi" & echo "Hi2"  
Hi2  
Hi
```

Zwei Commands nacheinander mittels && trennen, gleichzeitig mittels &

- Glob

```
cat *.txt      file1.txt file2.tex a.txt  
cat ?.txt     file1.txt file2.tex a.txt
```

Platzhalter für einen oder mehrere Zeichen in Dateinamen.

# III.

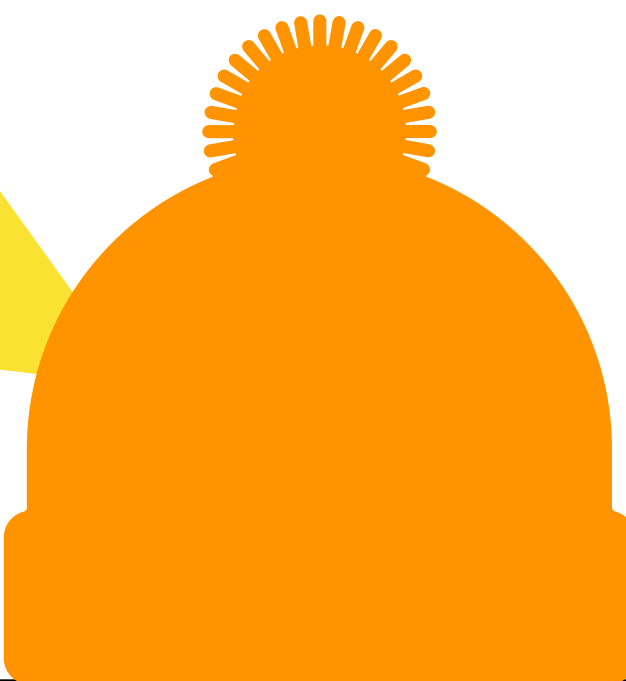
# Reguläre Ausdrücke

Ja, die kennt man aus TI, hier beziehen wir uns aber auf die typische Umsetzung in Programmiersprachen, *Perl Compatible Regular Expressions (PCRE)*

# Reguläre Ausdrücke

- Leistungsstarke und *einfache* Art Zeichenketten zu verarbeiten
- In Python durch das Paket `re` umgesetzt.
- Nicht nur in Python unterstützt, auch z.B. in Java mit `String.matches()` oder in PHP mit `preg_match()`

„Mit dem Tool <https://regexpr.com/> kann man seine eigenen RegEx schnell erstellen und ausprobieren.“



# Idee: RegEx

- (Teil-)Zeichenkette mit bestimmten Muster erkennen
  - „Hello“ oder „Hallo“ → "H(e|a)llo"
- Teile extrahieren
  - „Ratzeburger Allee 124“ → ".\* Allee (\d+)"
    - Capturing Group → "124"

Warum wird hier das Escape-Symbol benutzt?

# Syntax I: RegEx

|                 |                                    |   |
|-----------------|------------------------------------|---|
| Anfang und Ende | ^                                  | Anfang der Zeichenkette                             |
|                 | \$                                 | Ende der Zeichenkette                               |
| Escape          | \                                  | Escape-Symbol                                       |
| Zeichenmengen   | .                                  | Ein beliebiges Zeichen (keine neue Zeile)           |
|                 | [A-Z], [abdc], [0-9] oder [a-z0-9] | Menge von Zeichen                                   |
|                 | [^A-z], [^+ \ - ()]                | Negation einer Menge von Zeichen                    |
|                 | \w, \d, \s                         | Zeichen, Ziffer oder Leerzeichen (alle Arten)       |
|                 | \W, \D, \S                         | Nicht Zeichen, Ziffer oder Leerzeichen (alle Arten) |
| Wiederholungen  | *                                  | Beliebig oft des Zeichens/ Gruppe zuvor             |
|                 | +                                  | Mindest ein Mal das Zeichen/ Gruppe zuvor           |
|                 | ?                                  | Ein oder kein Mal das Zeichen/ Gruppe zuvor         |
|                 | {m}, {m,n}                         | m Mal bzw. m-n Mal das Zeichen/ Gruppe zuvor        |

# RegEx: Anfang, Ende, Zeichen

```
import re
```

```
re.match(r'^https?://www.example.com/?', "https://www.example.com/")
```

```
# ✓
```

```
re.match(r'^https?://www.example.com/?', "http://www.example.com/")
```

```
# ✓
```

```
re.match(r'^https?://www.example.com/?', "http://www0example0com/")
```

```
# ✓ (Aber das soll ja nicht so!)
```

```
re.match(r'^https?://www\.example\.com/?', "http://www0example0com/")
```

```
# ✗
```

```
re.match(r'^https?://www\.example\.com/?', "http://www.example.com/hallo")
```

```
# ✓ (Aber das soll ja nicht so!)
```

```
re.match(r'^https?://www\.example\.com/?$', "http://www.example.com/hallo")
```

```
# ✗
```

# RegEx: Wiederholungen und Mengen

```
pattern = r'^@[a-z]\.?uni-luebeck\.de'  
re.match(pattern, "bender@ifis.uni-luebeck.de")  
# ✗  
re.match(r'^@[a-z]+\.?uni-luebeck\.de', "bender@ifis.uni-luebeck.de")  
# ✓  
  
re.match(r'^@[a-z]+\.?uni-luebeck\.de', "m.bender@uni-luebeck.de")  
# ✗  
re.match(r'^@[a-z]*\.?uni-luebeck\.de', "m.bender@uni-luebeck.de")  
# ✓  
  
re.match(r'\w+@\w*\.?uni-luebeck\.de', "bender@ifis.uni-luebeck.de")  
# ✓  
  
re.match(r'\w+@\w*\.?uni-luebeck\.de', "m.bender@uni-luebeck.de")  
# ✗ (Achtung, \w beinhaltet kein Symbole!)  
  
re.match(r'\w+@\w*\.?uni-luebeck\.de', "bender@FAKEuni-luebeck.de")  
# ✓ (Sollte aber nicht)
```

Bei E-Mail-Adressen gibt es auch einige „abenteuerliche“ Formen ([RFC 3696](#)), daher besser eine Bibliothek nutzen.



# Syntax II: RegEx

|                 |           |   |
|-----------------|-----------|---|
| Oder            | ...   ... | Passt, falls einer der beiden RegEx ... passt |
| Gruppe          | ( ... )   | Capturing Group für RegEx ...                 |
|                 | (?: ... ) | Non-Capturing Group RegEx ...                 |
|                 | (?! ... ) | Passt, falls der RegEx ... nicht passt        |
|                 | usw.      | Es gibt noch viele weitere Gruppen            |
| Weitere Zeichen | \n        | Zeilenumbruch (neue Zeile)                    |
|                 | \r        | Carriage return                               |
|                 | \t        | Tabulator                                     |

# RegEx: Gruppen

```
import re
```

```
m = re.match(r'^H(e|a)llo$', "Hello")  
print(m.groups()) # ('e',)
```

```
m = re.search(r'H(e|a)llo', "Hallo Magnus, Hello Mr. Bender,")  
print(m.groups()) # ('a',)
```

```
m = re.findall(r'H(e|a)llo', "Hallo Magnus, Hello Mr. Bender,")  
print(m) # ['a', 'e']
```

```
m = re.findall(r'(H(e|a)llo)', "Hallo Magnus, Hello Mr. Bender,")  
print(m) # [('Hallo', 'a'), ('Hello', 'e')]
```

```
m = re.findall(r'H(?:e|a)llo ([^,]+)', "Hallo Magnus, Hello Mr. Bender,")  
print(m) # ['Magnus', 'Mr. Bender']
```

```
mail = re.compile(r'^@[^@]+(?:[a-z]*\.)?uni-luebeck\.de')  
print(mail.match("m.bender@uni-luebeck.de")) # ✓  
print(mail.match("bender@ifis.uni-luebeck.de")) # ✓  
print(mail.match("bender@FAKEuni-luebeck.de")) # ✗
```

Mehrzeilige Strings können mit `"""` erstellt werden.

# REGEX. GRUPPEN UND ERSETZUNGEN

```
import re
```

```
program = """
```

```
class Zug:
```

```
    def __ini__(self, a, b, c):
```

```
        pass
```

```
    def _internal(self):
```

```
        pass
```

```
class Lokomotive():
```

```
"""
```

Sucht erstes Vorkommen

```
m = re.search(r'class ([a-zA-Z_][0-z]*)?(:\(\))?:', program)
```

```
print(m.group(0), m.group(1))
```

Sucht alle Vorkommen

```
print(re.findall(r'class ([a-zA-Z_][0-z]*)?(:\(\))?:', program))
```

```
print(re.findall(r'class ([a-zA-Z_][0-z]*)(\(\))?:', program))
```

```
print(re.sub(
```

```
    r'def __ini__\((self[^\)]*)\):',
```

```
    r'def __init__(\1):',
```

```
    program
```

```
))
```

```
    def __init__(self, a, b, c):
```

Nimmt man für den zweiten Parameter `""`, dann löscht man alles, auf das der RegEx passt.

Suchen und ersetzen, beim Ersetzen können Gruppen eingesetzt werden.

```
$> python3 name.py
```

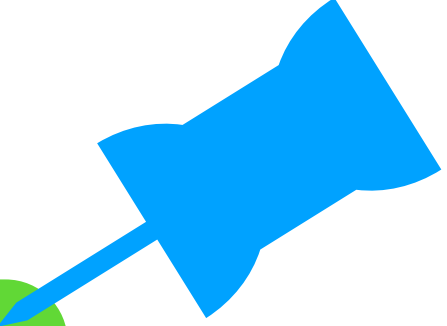
```
class Zug: Zug
```

```
['Zug', 'Lokomotive']
```

```
[('Zug', ''), ('Lokomotive', '()')]
```

# Zusammenfassung

- Lösungsmöglichkeiten Aufgabe 1
- Notebooks lokal und in der Cloud
- Grundlagen von Bash
- Reguläre Ausdrücke



Wir haben jetzt  
Programmcode, Dokumente  
und Notebooks.

Nun müssen wir das alles  
verwalten und organisieren!



~~Heute~~

# Inhaltsübersicht

1. Programmiersprache Python
  - a) *Einführung, Erste Schritte*
  - b) *Grundlagen*
  - c) *Fortgeschritten*
2. Auszeichnungssprachen
  - a) *LaTeX, Markdown*
3. Benutzeroberflächen und Entwicklungsumgebungen
  - a) *Jupyter Notebooks lokal und in der Cloud (Google Colab)*
4. Versionsverwaltung
  - a) **Git, GitHub**
5. Wissenschaftliches Rechnen
  - a) NumPy, SciPy
6. Datenverarbeitung und -visualisierung
  - a) Pandas, matplotlib, NLTK
7. Machine Learning (scikit-learn)
  - a) Grundlegende Ansätze (Datensätze, Auswertung)
  - b) Einfache Verfahren (Clustering, ...)
8. DeepLearning
  - a) TensorFlow, PyTorch, HuggingFace Transformers