

# **Vorlesung Ingenieurinformatik**

Lukas Arnold      Simone Arnold      Florian Bagemihl  
Matthias Baitsch      Marc Fehr      Maik Poetzsch  
Sebastian Seipel

2026-02-25

# **Table of contents**

# **Übersicht**

# **Part I**

## **w-pseudocode**

# 1 Werkzeugbaustein Pseudocode



Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch und Sebastian Seipel. Werkzeugbaustein Pseudocode von Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar auf [GitHub](#). Ausgenommen von der Lizenz sind alle Logos und anders gekennzeichneten Inhalte. 2025

Zitiervorschlag

Arnold, Lukas, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch, und Sebastian Seipel. 2025. „Bausteine Computergestützter Datenanalyse. Werkzeugbaustein Pseudocode“. <https://github.com/bausteine-der-datenanalyse/w-pseudocode>.

BibTeX-Vorlage

```
@misc{BCD-pseudocode-2025,  
  title={Bausteine Computergestützter Datenanalyse. Werkzeugbaustein Pseudocode},  
  author={Arnold, Lukas and Arnold, Simone and Bagemihl, Florian and Baitsch, Matthias and Fehr, Marc and Poetzsch, Maik and Seipel, Sebastian},  
  year={2025},  
  url={https://github.com/bausteine-der-datenanalyse/w-pseudocode}}
```

## 2 Voraussetzungen

Für die Bearbeitung dieses Bausteins bestehen keine Voraussetzungen. Die Bearbeitungszeit beträgt je nach Umfang der bearbeiteten Übungsaufgaben circa 120 bis 300 Minuten.

Während der Bearbeitung des Bausteins *können* Flussdiagramme unter anderem mit Python oder R erstellt werden. Dazu werden folgende Bibliotheken benötigt:

- Python: [Schemdraw](#) und/oder [graphviz](#). Zur Ausführung von Graphviz wird außerdem eine lokale Installation des Programms [Graphviz](#) benötigt.
- R: [DiagrammeR](#)

Am Ende des Bausteins besteht die Möglichkeit, ein Programm zur Überprüfung einer Börsenstrategie zu entwickeln. Zur Veranschaulichung wird ein Auszug aus der von Robert Shiller erstellten Datei `ie_data.xls` mit historischen Börsenkursen des S&P 500 verwendet. Die Datei kann unter <https://shillerdata.com/> bezogen werden.

## 3 Lernziele

In diesem Baustein lernen Sie, mit Hilfe von Pseudocode (komplexe) Aufgabenstellungen zu strukturieren und Schritt für Schritt ein Programm für die computergestützte Datenanalyse zu entwickeln. Außerdem lernen Sie, den Programmablauf für das eigene Verständnis und den Austausch mit Dritten in Form eines Flussdiagramms zu visualisieren.

Wenn Sie dieses Modul durchgearbeitet haben, können Sie ...

- die Entwicklung von Pseudocode als Kreativitätstechnik einsetzen,
- ein Programm in Pseudocode beschreiben,
- ein Programm in abgegrenzte Zwischenschritte aufteilen und diese in eine zweckmäßige Reihenfolge bringen,
- die Abhängigkeiten zwischen den Zwischenschritten beschreiben,
- die Voraussetzungen für die Ausführung jedes Zwischenschritts und des gesamten Programms benennen sowie
- den Programmablauf als Flussdiagramm visualisieren.

Schlagworte: Ideen entwickeln, Programmbeschreibung, Pseudocode, Programmablaufplan, DIN 66001

## 4 Pseudocode

### 4.1 Was ist Pseudocode?



Figure 4.1:

Toast Dining Eating von OpenClipart-Vectors ist lizenziert unter [Pixabay Content License](#). Das Werk ist abrufbar auf [Pixabay](#). 2013

Denken Sie an die Zubereitung eines Festtagsmahls. Sie haben im Internet ein Rezept gefunden, alle Zutaten von der Zutatenliste eingekauft und beginnen entsprechend der angegebenen Zubereitungszeit vor der Ankunft Ihrer Gäste zu kochen. Klingt gut, oder? Nun, dieses Vorgehen verspricht hungrig wartende Gäste, denn Rezepte sind häufig keine fertige Schritt-für-Schritt-Anleitung, sondern erfordern einige Vorarbeit:

- Mal fehlen Zutaten auf der Zutatenliste, mal bleibt eine zu besorgnde Zutat im Rezept unerwähnt und am Ende übrig. Im vierten Schritt ruht der Teig bereits seit zwei Stunden im Kühlschrank und der Ofen ist schon auf 250 Grad vorgeheizt. Eine vollständige Aufgabenbeschreibung erfordert Vorbereitung: Nur wer alle Abhängigkeiten kennt, wird pünktlich fertig.

- In der Küche gilt eine eigene Fachsprache: Wissen Sie, was blanchieren bedeutet, wieso selbst sauberes Gemüse geputzt werden soll, wann Nudeln al dente sind oder ob die Soße erst lichterloh brennen sollte, damit man sie ablöschen kann? Falls nicht, ist zunächst etwas Übersetzungsarbeit in eine für Sie verständliche Sprache gefragt.
- Wer kein Messer hat, kann kein Brot schneiden: Die Zutaten sind nicht alles, zu einem gelungenen Gericht gehört auch das richtige Werkzeug. Haben Sie alles in der Küche, was Sie zur Zubereitung brauchen?

Kurz: Sie brauchen einen eigenen Plan – nicht nur beim Kochen, sondern auch bei der computergestützten Datenanalyse. Denn auch die Datenanalyse erfolgt selten linear. Stattdessen wechseln sich Schritte zur Organisation, Visualisierung, Auswertung und Weiterverarbeitung von Daten ab. Dazu steht eine Vielzahl unterschiedlicher Programme mit eigener Syntax zur Auswahl. Oder ist eine selbst geschriebene Funktion besser geeignet? Häufig liegen Daten in unterschiedlichen Formaten vor und müssen für die Verwendung speziälizierter Programme transformiert werden. Dies sind typische Herausforderungen der Datenanalyse, die durch eine gute Planung aufgefangen werden können. Bei der Planung hilft Ihnen Pseudocode.

Pseudocode beschreibt ein Programm und informatische Konzepte in einer leicht verständlichen, alltäglichen Sprache. Die Entwicklung von Pseudocode kann sowohl als Kreativitätstechnik als auch als Schablone zur strukturierten Lösung von Aufgabenstellungen genutzt werden. Pseudocode erlaubt es, sich zunächst auf die konzeptionelle Lösung einer Aufgabenstellung und die Planung des Programmaufbaus zu fokussieren. Dies kann sinnvoll sein, um durch die Unterteilung des Programms in Zwischenschritte die Entwicklung zu vereinfachen, benötigte Programme, Pakete und Methoden zu identifizieren oder den Arbeitsaufwand abzuschätzen.

Pseudocode ist auch ein Instrument, um sich mit Dritten über ein Programm auszutauschen, zum Beispiel um ein Problem im Code zu erörtern. Denn Pseudocode setzt keine Kenntnis einer bestimmten Programmiersprache voraus und ist intuitiv verständlich. Zur Unterstützung der Kommunikationsfunktion von Pseudocode kann dieser zum Beispiel als Flussdiagramm visualisiert werden.

### **! Important ??: Pseudocode**

Pseudocode beschreibt einen Lösungsweg für informatische Aufgabenstellungen in formalisierter Alltagssprache statt mit den Ausdrücken und der Syntax einer Programmiersprache. Das Präfix pseudo stammt aus dem Griechischen und bedeutet falsch oder nur so aussehen als ob. Pseudocode ist ‚falscher‘ Programmcode, der mit natürlicher Sprache gebildet wird. [@Pseudocode-was-ist] Pseudocode ist darüber hinaus auch ein Kommunikationsmittel, um sich mit anderen über informatische Probleme und deren Lösung auszutauschen (z. B. Kommiliton:innen, Betreuer:innen).

## **4.2 Pseudocode von einfach bis ausführlich**

In welcher Form und wie detailliert Sie Pseudocode formulieren, ist abhängig von Ihrem Kenntnisstand und Ihrem Ziel: Für die Strukturierung Ihres Programms in aufeinanderfolgende Zwischenschritte und die Identifizierung von Abhängigkeiten ist eine einfache Auflistung von Arbeitsschritten ausreichend. Für die Entwicklung eines Algorithmus sowie die Identifizierung von benötigten Methoden (z. B. Schleifen) und Werkzeugen (z. B. spezialisierte Pakete) wird es erforderlich sein, Programmanweisungen detaillierter zu beschreiben. Eine ausführliche Darstellung hilft Ihnen zum Beispiel, wiederkehrende Arbeitsschritte zu identifizieren, die in eine Funktion ausgelagert werden können. Für die Präsentation Ihres Programms auf einer Fachtagung oder die Dokumentation in einem Benutzer:innenhandbuch kann ein Flussdiagramm zweckmäßig sein.

### **4.3 einfach**

Programmanweisung mit sprechendem Namen

**SortiereAufsteigend**

### **4.4 mittel**

Anweisungsblock in Alltagssprache

```
SortiereAufsteigend
  DurchlaufeDatensatz (LängeDatensatz - 1) mal # Anzahl Wiederholungen
    Von Anfang bis (LängeDatensatz - 1) # Paarweise Vergleiche
      Vergleiche Wert und Nachfolger
      WENN Wert größer als Nachfolger DANN
        Vertausche Wert und Nachfolger
      AusgabeDatensatz
```

### **4.5 ausführlich**

Anweisungsblock im Stil und mit Begriffen der Programmierung

```
SortiereAufsteigend
  WENN LängeDatensatz > 1 DANN
    VON Datensatz[IndexA = 1] BIS Datensatz [IndexA = LängeDatensatz - 1] TUE # Anzahl Schritte
      VON Datensatz[IndexB = 1] BIS Datensatz [IndexB = LängeDatensatz - 1] TUE # Paarweise
        WENN Datensatz[IndexB] > Datensatz[IndexB + 1] DANN
          Vertausche(Datensatz[IndexB], Datensatz[IndexB + 1])
          Schreibe(Datensatz[IndexB], nach = Zwischenlager)
          Schreibe(Datensatz[IndexB + 1], nach = Datensatz[IndexB])
          Schreibe(Zwischenlager, nach = Datensatz[IndexB + 1])
        WENN IndexB < LängeDatensatz - 1 DANN
          Erhöhe IndexB
        WENN IndexA < LängeDatensatz - 1 DANN
          Erhöhe IndexA
      AusgabeDatensatz
```



## 4.6 Flussdiagramm

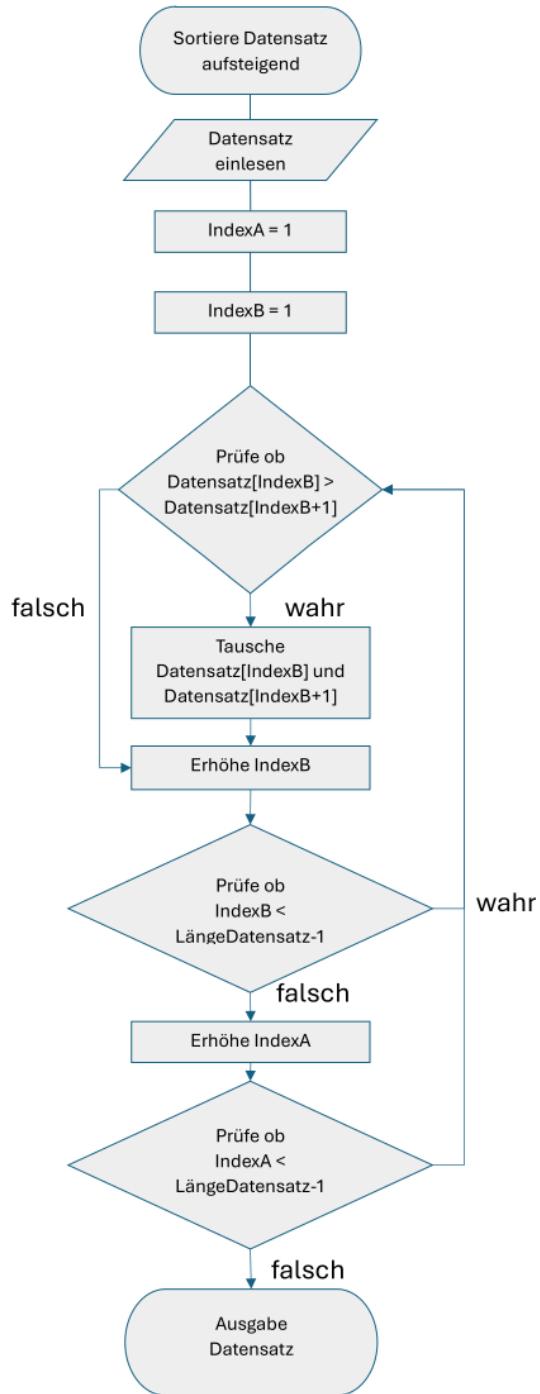


Figure 4.2: Flussdiagramm Sortiere Aufsteigend

Flussdiagramm von Marc Sönnecken und Maik Poetzsch

# 5 Pseudocode erstellen

Das im Folgenden vorgestellte Schema unterstützt Sie dabei, Ideen zu entwickeln, ein Programm zu beschreiben und den Programmablauf zu visualisieren. Pseudocode hilft Ihnen, Ihr Programm sowie die benötigten Werkzeuge und Methoden in einer für Sie und Dritte verständlichen Weise zu beschreiben. Wie Sie Pseudocode verfassen, ist eine individuelle Angelegenheit, denn für die Erstellung von Pseudocode gibt es nur wenige Regeln. Pseudocode soll vor allem Ihnen helfen, einen Lösungsweg für eine Aufgabenstellung zu entwickeln. Pseudocode ist dann ‚richtig‘ geschrieben, wenn er Ihnen hilft:

- Ihre Gedanken zu fokussieren,
- eine Aufgabenstellung für Sie nachvollziehbar zu beschreiben,
- die Aufgabenstellung in abgegrenzte Teilaufgaben zu untergliedern,
- für die Teilaufgaben eine Lösungsvorschrift (Algorithmus) zu entwickeln,
- benötigte Methoden und Werkzeuge zu identifizieren,
- Ihre Lösungsvorschriften als ein Programm aufeinanderfolgender Arbeitsschritte zu beschreiben,
- den Zeitaufwand für die Umsetzung abzuschätzen sowie
- sich mit Dritten über Ihr Programm auszutauschen.

## 5.1 Übungsaufgaben

Für die Bearbeitung dieses Bausteins stehen zwei Übungsaufgaben zur Auswahl. Die Übungsaufgaben sprechen auch Studierende ohne Vorkenntnisse in der Datenanalyse an und verdeutlichen Teilaspekte der Programmierung. Am besten eignet sich jedoch ein eigenes Projekt, das Sie bearbeiten möchten. Dazu haben Sie im Reiter ‚Eigenes Projekt‘ die Möglichkeit, eine eigene Aufgabenstellung zu formulieren.

## 5.2 Übersicht

- ohne Vorkenntnisse Datenanalyse: Hefezopf backen  
Lernziele: Aufgabenstellung in aufeinanderfolgende Arbeitsschritte untergliedern, Arbeitsschritte vollständig beschreiben, Abhängigkeiten und Voraussetzungen identifizieren (benötigte Werkzeuge identifizieren, Zeitaufwand schätzen)
- mit Vorkenntnissen Datenanalyse: Vitamin C bei Meerschweinchen  
Lernziele: Datenanalyse in Pseudocode beschreiben.

## 5.3 Eigenes Projekt

Hier können Sie die Aufgabenstellung bzw. das Ziel Ihres Projekts festhalten. Bitte beachten Sie, dass Ihr Text nicht gespeichert wird.

## 5.4 Hefezopf backen

Ihre Freundin Lisa schickt Ihnen ein Rezept, dass Sie im Internet gefunden hat. Sie schreibt, dass Sie morgen schon eine Stunde früher zum geplanten Treffen kommen kann, um vorher gemeinsam mit Ihnen zu backen – zu zweit würde man schon schneller fertig werden als in den im Rezept angegebenen 65 Minuten. Lisa fragt auch, ob Sie noch Zutaten mitbringen soll.

**Wie antworten Sie Lisa? Modellieren Sie den Backprozess.**

Das folgende Rezept wurde von Anna-Lena erstellt und ist abrufbar unter <https://www.einfachbacken.de/rezepte/hefezopf>.

### Zarter Hefezopf

Arbeitszeit 40 Min.

Backen 25 Min.

---

#### Zutaten

---

250 ml	Milch
475 g	Weizenmehl (Type 405)
60 g	Zucker
½ Würfel	frische Hefe (ca. 21 g)
50 g	weiche Butter (Zimmertemperatur)
1 Prise	Salz
1	Ei (Gr. M)
	etwas Milch zum Bestreichen
	etwas Hagelzucker zum Bestreuen

---

## Zutaten

---

etwas Mehl zur Teigverarbeitung

---

### 1. Schritt

**250 ml** Milch, **475 g** Weizenmehl (Type 405), **1 Prise** Zucker, **½ Würfel** frische Hefe (ca. 21 g)

Milch erwärmen bis sie lauwarm ist. Mehl in eine Schüssel sieben. Eine Mulde darin bilden und die Hefe in die Mulde bröseln. 3 EL von der lauwarmen Milch mit 1 Prise Zucker vermischen und über die Hefe in der Mulde gießen. Mit einem Löffel die Hefe-Milchmischung etwas vermischen (noch nicht das Mehl einkneten). Die Schüssel mit einem Geschirrhandtuch abdecken und an einem warmen Ort ca. 15 Min. gehen lassen.

### 2. Schritt

**1 Ei** (Gr. M), **60 g** Zucker, **1 Prise** Salz, **50 g** weiche Butter (Zimmertemperatur)

Ei, restliche Milch, restlichen Zucker und Salz in die Schüssel geben und zusammen mit der Hefemischung und dem Mehl 3 Min. auf niedriger Stufe, dann ca. 5 Min. auf hoher Stufe mit den Knethaken des Rührgeräts verkneten. Butter in Stücke nach und nach unterkneten. Damit der Teig später gut aufgeht, sollte der Teig mindestens 5 Min. kräftig geknetet werden. Sonst kann der Teig später zusammenfallen oder klebrig sein!

### 3. Schritt

etwas Mehl zur Teigverarbeitung

Schüssel mit dem Teig nochmals mit einem Geschirrhandtuch abdecken und weitere 60 Min. an einem warmen Ort gehen lassen. Dann den[ ]Teig auf eine bemahlte Arbeitsfläche geben und in drei Teile teilen. Die Teigstücke jeweils zu einer langen Wurst mit 40 cm Länge rollen. Teigsträhnen zu einem Zopf flechten. Die Enden miteinander verdrehen und unter den Zopf legen, damit sie einen schönen Abschluss bilden. Zopf auf ein mit Backpapier belegtes Blech legen und mit einem Geschirrhandtuch abdecken. Nochmals 45 Min. gehen lassen.

### 4. Schritt

etwas Milch zum Bestreichen, etwas Hagelzucker zum Bestreuen

Währenddessen den Backofen auf **200 Grad Ober-/ Unterhitze (Umluft: 180 Grad)** vorheizen. Zopf mit etwas Milch bestreichen und mit Hagelzucker bestreuen. Zopf schließlich im vorgeheizten Ofen **ca. 15-20 Minuten** leicht bräunlich backen. Vollständig auskühlen lassen. Der Zopf kann auch wunderbar eingefroren werden.

## 5.5 Vitamin C bei Meerschweinchen

In einer Gruppe von 60 Meerschweinchen wurde die Länge der zahnbildenden Zellen (Odontoblasten) in Micron gemessen (**len**). Den Tieren wurde zuvor Vitamin C in Form von Ascorbinsäure (VC) oder Orangensaft (OJ) verabreicht (**supp**). Die Meerschweinchen erhielten Dosen von 0.5, 1 oder 2 Milligramm Vitamin C pro Tag (**dose**). [@Crampton.1947]

Welche Wirkung hat Vitamin C auf das Zahnwachstum von Meerschweinchen? Erläutern Sie Ihr Vorgehen mit Hilfe von Pseudocode oder mit Hilfe eines Flussdiagramms, um den Effekt der Verabreichungsart und der Dosis zu bestimmen.

#	len	supp	dose
1	4.2	VC	0.5
11	16.5	VC	1
21	23.6	VC	2
31	15.2	OJ	0.5
41	19.7	OJ	1
51	25.5	OJ	2

Wenn Sie sich den vollständigen Datensatz ansehen möchten, können Sie diesen in R mit `ToothGrowth` aufrufen oder den Datensatz hier herunterladen: [ToothGrowth.csv](#)

## 5.6 Mit EVA zum Fokusprint



Figure 5.1:

Adam Bible Nature von CCXpistavos ist lizenziert unter [Pixabay Content License](#). Das Werk ist abrufbar auf [Pixabay](#). Die Sprechblase wurde ergänzt.

Manchmal ist aller Anfang schwer. Der erste Schritt zur Programmentwicklung besteht in der Ideensammlung: Was soll genau getan werden, welche Schritte sind dafür erforderlich, was kann bereits erledigt werden, was muss noch vorbereitet oder recherchiert werden? **Wenn Sie bereits eine gute Vorstellung von der Aufgabenlösung haben, können Sie diesen Schritt überspringen.**

Der **Fokusprint** (Scheuermann 2016) ist eine schnelle Schreibdenkübung, um einen Einstieg ins Schreiben zu einem bestimmten Thema zu finden. Die Übung kann als Kreativitätstechnik für den Einstieg in ein Thema, aber auch zwischendurch als Denkhilfe eingesetzt werden. Es geht dabei darum, mit einem hohen Schreibtempo drauflos zu schreiben. Dadurch schreiben Sie nahe an Ihrer inneren Sprache und Ihre Kreativität kann freien Lauf nehmen. [@Scheuermann2016, 74, 78]

Der Fokusprint wird in zwei Schritten durchgeführt: Der erste Schritt besteht in einer fünfminütigen Schreibphase. Formulieren Sie auf einem Blatt Papier oder am Computer die Aufgabenstellung oder das Problem, zu dem Sie Ihre Gedanken sammeln möchten. Hier können Sie sich mit einem Stichwort kurzfassen oder eine konkrete Frage formulieren. Stellen Sie sich einen Wecker auf 5 Minuten und beginnen mit dem Schnellschreiben auf dem vorbereiteten Blatt. Es gibt nur eine Regel: Wenn Sie bemerken, dass Ihre Gedanken vom Thema der Überschrift abschweifen, besinnen Sie sich auf das Thema, zum Beispiel, indem Sie die Aufgabenstellung erneut aufschreiben (einfach dort, wo Sie gerade schreiben). [@Scheuermann2016, 78]

## **Fokusprint**

**In diesem Textfeld können Sie Ihren Fokusprint durchführen. Bitte beachten Sie, dass Ihr Text nicht gespeichert wird. Also dann: 3, 2, 1 – schreiben Sie in eigenen Worten los!**

Im zweiten Schritt erfolgt die Auswertung. Lesen Sie Ihren Fokusprint durch und markieren Sie, was Ihnen wichtig erscheint. Können Sie Zwischenschritte identifizieren? Wenn ja, markieren Sie diese und fügen Ergänzungen, Kommentare oder Fragen hinzu. [vgl. @Scheuermann2016, 78]

### Tip ??: Zwischenschritte identifizieren

Zwischenschritte teilen eine komplexe Aufgabenstellung in überschaubare Arbeitspakete auf, die jeweils eine bestimmte Funktion im Programmablauf erfüllen (z. B. das Sortieren von Daten) oder ein bestimmtes Arbeitsergebnis erzeugen (z. B. eine Grafik). Ein Zwischenschritt ist also ein Unterprogramm und bei der Bildung von Zwischenschritten geht es somit darum, eine Aufgabenstellung in Teilaufgaben aufzuteilen und eine Vorstellung davon zu gewinnen, welche Arbeitsschritte in einem Unterprogramm ausgeführt werden müssen, um die Teilaufgabe zu lösen.

Die Bildung von Zwischenschritten dient vor allem der Komplexitätsreduktion. Ein zur Lösung einer abgegrenzten Teilaufgabe entwickeltes Unterprogramm hat zum einen den Vorteil, dass es leicht getestet werden kann, um Fehler zu finden und zu beheben. Zum anderen kann ein Unterprogramm wiederholt im Programmablauf aufgerufen werden, z. B. um die Lösung einer anderen, komplexeren Teilaufgabe zu vereinfachen.

Bei der Abgrenzung von Zwischenschritten können auch andere Kriterien als die erbrachte Funktion oder das produzierte Arbeitsergebnis sinnvoll sein, etwa nach dem Ort der Datenverarbeitung (z. B. an der Messstelle, am Arbeitsrechner, im Rechenzentrum) oder nach den verwendeten Werkzeugen (z. B. Mikrocontroller, Python, C++, manuelle Datenverarbeitung).

#### **Tip 5.1: Mögliche Zwischenschritte für die Übungsaufgaben**

Hefezopf backen

- Unterteilung nach Funktion: Vorbereiten der Zutaten, Verarbeiten der Zutaten, Backen
- Unterteilung nach Arbeitsergebnissen: Milchmischung zubereiten, Hefemischung zubereiten, Rohteig zubereiten, Teigzopf formen, Zopf backen

Vitamin C bei Meerschweinchen

- Unterteilung nach Funktion: Teildatensätze bilden, Datensatz auswerten, Datensatz darstellen, Flussdiagramm erstellen
- Unterteilung nach Arbeitsergebnis: tabellierter Mittelwertvergleich, Boxplot für alle Teilgruppen, Mermaid Flussdiagramm

Das **EVA-Prinzip** hilft Ihnen, Zwischenschritte vollständig zu beschreiben. Das EVA-Prinzip ist ein Grundmuster der computergestützten Datenverarbeitung und steht für **Eingabe**, **Verarbeitung** und **Ausgabe**. Diese Schritte folgen aufeinander: Zuerst werden die Daten erfasst, dann erfolgt die Datenverarbeitung, zuletzt werden die Ergebnisse ausgegeben.

- Eingabe: Welche Daten liegen als Eingabe vor? Welches Format haben die Daten?

- Verarbeitung: Welche Arbeitsschritte müssen durchgeführt werden, um die beschriebene Ausgabe zu erreichen?
- Ausgabe: Welches Ergebnis soll durch die Datenverarbeitung erzeugt werden? Welches Format hat die Ausgabe?

Fassen Sie zum Abschluss Ihren Fokusprint in einem Kernsatz zusammen, der das für Sie Wichtigste hervorhebt. Dies kann eine Feststellung, aber auch eine offene Frage, die Sie weiter verfolgen möchten, sein. Markieren Sie diesen Kernsatz zusätzlich. [@Scheuermann2016, 79]

Damit haben Sie den ersten Schritt zur Formulierung einer vollständigen Programmbeschreibung geschafft! Wenn Sie möchten, können Sie Ihre Gedanken zu einer noch offenen Frage oder zu einem Zwischenschritt mit einem erneuten Fokusprint vertiefen. Andernfalls folgt nun der nächste Schritt.

## 5.7 Programmbeschreibung in Pseudocode

Sie haben nun eine Vorstellung von Ihrem Lösungsweg, den Zwischenschritten, aus denen dieser besteht, und von ihrer Abfolge. Im zweiten Schritt wird Ihr Programm durch die Beschreibung mit Pseudocode formalisiert. Dazu werden Programmanweisungen zwar alltagssprachlich, aber im Stil und mit Begriffen der Programmierung formuliert. Dies bedeutet,

1. für Programmanweisungen sprechende Namen zu vergeben. In der Datenanalyse benutzte Anweisungen sind:
  - Daten importieren (import), z. B. HoleDatenVonOrt
  - Daten organisieren (tidy), z. B. SortiereAufsteigend
  - Daten transformieren (transform), z. B. BildeDurchschnitt
  - Daten visualisieren (visualise), z. B. ErzeugeHistogramm
  - Daten modellieren (model), z. B. ErzeugeLinearesModell
  - Daten exportieren (export), z. B. SpeicherePlot
- [vgl. @R-for-Data-Science, Kapitel Whole game]
2. den Programmablauf zu dokumentieren, indem
  - aufeinanderfolgende Programmanweisungen untereinander geschrieben,
  - Blöcke von Programmanweisungen durch Einrückung und/oder Einklammerung kenntlich gemacht und
  - zusammenhängende Programmanweisungen in abgegrenzten Zwischenschritten gruppiert werden.

3. Programmanweisungen von erläuternden Teilen durch Kommentare zu trennen, indem
  - Kommentare durch Sonderzeichen gekennzeichnet, beispielsweise `// Kommentar`, `# Kommentar`, `%% Kommentar` oder `/* Kommentar */`, und
  - einzelnen Arbeitsschritten vorangestellt (z. B. Kurzbeschreibung von Zwischen-schritten nach EVA) und/oder innerhalb einer Zeile benutzt werden.

4. Komplexe Programmanweisungen mit informatischen Begriffen auszudrücken, ggf. in Anlehnung an die Begriffe und Syntax der von Ihnen gewählten Programmiersprache. Dies umfasst:

**Fallunterscheidungen:** Fallunterscheidungen machen die Ausführung von Programmanweisungen abhängig von einer oder von mehreren Bedingungen.

- WENN A (kleiner, kleiner gleich, genau gleich, größer als, größer gleich, ungleich) B, DANN C, SONST D
- WENN A B1 UND B2, DANN C
- WENN A B1 ODER B2, DANN C

**Schleifen:** Schleifen wiederholen Programmanweisungen solange die Eintrittsbedingung gilt bzw. die Abbruchbedingung eintritt.

- SOLANGE A, TUE C
- VON A BIS B, TUE C

**Funktionen:** Funktionen bündeln Programmanweisungen, damit Programmteile mehrfach verwendet werden können. Funktionen sind eine Form von Unterprogrammen.

- Funktion `TueXY(Argument 1, Argument 2, ...)`
- Anweisung1
- Anweisung2
- ...

#### **i** Note ??: Programmbeschreibung in Pseudocode

```
# Eingabe: eindimensionaler, ordinaler Datensatz
# Verarbeitung: Prüfung ob Datensatz mindestens zwei Werte enthält
## falls nein: Fehlermeldung
## falls ja: aufsteigende Sortierung mit Bubblesort mit Kontrollstruktur für den äußeren S
# Ausgabe: aufsteigend sortierter Datensatz

SortiereAufsteigend(Datensatz)
  WENN LängeDatensatz > 1 DANN
```

```

# äußere Schleife
do_work = WAHR # Kontrollstruktur für die äußere Schleife
WENN do_work == WAHR DANN

VON Datensatz[IndexA = 1] BIS Datensatz [IndexA = LängeDatensatz - 1] TUE
    do_work = FALSCH

    # innere Schleife für paarweise Vergleiche
    VON Datensatz[IndexB = 1] BIS Datensatz [IndexB = LängeDatensatz - 1] TUE
        WENN Datensatz[IndexB] > Datensatz[IndexB + 1] DANN
            Vertausche(Datensatz[IndexB], Datensatz[IndexB + 1])
            Schreibe(Datensatz[IndexB], nach = Zwischenlager)
            Schreibe(Datensatz[IndexB + 1], nach = Datensatz[IndexB])
            Schreibe(Zwischenlager, nach = Datensatz[IndexB + 1])
            do_work = WAHR # merke, wenn etwas vertauscht wurde
        WENN IndexB < LängeDatensatz - 1 DANN
            Erhöhe IndexB

        WENN IndexA < LängeDatensatz - 1 DANN
            Erhöhe IndexA

    AusgabeDatensatz
    SONST
        Melde("Der Datensatz muss mindestens zwei Elemente enthalten!")

```

*Hinweis: Die manuelle Erhöhung der Zählindizes ist bei Programmiersprachen in der Regel nicht erforderlich und dient nur der besseren Verständlichkeit.*

Mit der Formalisierung Ihres Programms in Pseudocode haben Sie eine intuitiv verständliche, vollständige Programmbeschreibung entwickelt. Im nächsten Schritt können Sie Ihr Programm grafisch darstellen.

## 5.8 Übungsaufgabe zählen

Beschreiben Sie ein Programm, das von 1-10 und von 15 bis 20 zählt, in Pseudocode.

In diesem Textfeld können Sie Ihren Pseudocode schreiben. Bitte beachten Sie, dass Ihr Text nicht gespeichert wird.

## 5.9 Musterlösung Pseudocode

```
i = 0
SOLANGE i kleiner als 10 TUE
    erhöhe i um 1
    Ausgabe i
i = 14
SOLANGE i kleiner als 20 TUE
    erhöhe i um 1
    Ausgabe i
```

### 5.9.1 Karel The Robot

Wenn Sie die Anwendung informatischer Konzepte wie Fallunterscheidung oder Schleifen üben möchten, ist [Karel The Robot](#) einen Blick wert. Das Programm führt mit einer eigenen Pseudocode-Sprache in die Anwendung informatischer Kontrollstrukturen ein.

Eine Vorstellung des Programms, Hinweise zur Installation und Bedienung finden Sie auf der im vorherigen Absatz verlinkten GitHub-Seite oder auf dem YouTube-Kanal der Medienberatung Niedersachsen:

[https://www.youtube.com/watch?v=vD8RN\\_WbvLo](https://www.youtube.com/watch?v=vD8RN_WbvLo)

Programmieren lernen mit Karel von Medienberatung Niedersachsen ist lizenziert unter [CC-BY](#). Das Werk ist abrufbar auf [YouTube](#). 2022

## 5.10 Programmablauf visualisieren



Figure 5.2:

Lead 1 von CocoMaterial ist lizenziert unter [CC0 1.0](#). Das Werk ist abrufbar auf [CocoMaterial](#).

Eine anschauliche grafische Darstellung verhilft Pseudocode zu noch mehr Klarheit und unterstützt dadurch insbesondere den Austausch mit Dritten. Die grafische Darstellung eines Programms wird Programmablaufplan oder Flussdiagramm genannt. Die für einen Programmablaufplan verwendeten Symbole sind in der [[@DIN-66001-1983](#)] genormt. Die Formen werden auch als Knoten (nodes) bezeichnet, die Verbindungen als Kanten (edges).

[[@DIN-66001-1983](#)]

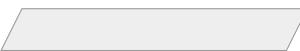
### **Warning ??: Darstellung von Daten nach DIN 66001**

Die DIN 66001 definiert auch Sinnbilder zur Darstellung von Daten. Die Norm regelt aber für Programmablaufpläne: "Daten werden nicht dargestellt. Siehe Beispiel in Abschnitt A.2." [[@DIN-66001-1983](#), S. 2]. Gleichwohl sind im Beispiel A.2 spezifische Typen von Daten dargestellt, beispielsweise manuell oder maschinell zu verarbeitende Daten [[@DIN-66001-1983](#), S. 13-14].

Jedoch sind die in den Nummern 6.2.2 bis 6.2.10 definierten Sinnbilder für spezifische Datentypen nicht in allen Diagrammwerkzeugen vollständig verfügbar. Dies betrifft:

- [Graphviz](#) (z. B. maschinell zu verarbeitende Daten entsprechend Nummer 6.2.2)
- [Mermaid](#) (z. B. maschinell zu verarbeitende Daten entsprechend Nummer 6.2.2)
- Bürossoftwarepaket LibreOffice (z. B. manuell zu verarbeitende Daten entsprechend Nummer 6.2.3)

In der Darstellung von Programmablaufplänen ist die Verwendung des Sinnbilds Nr. 6.2.1 Daten allgemein üblich.

 **Daten (Nummer 6.2.1)** Darstellung für Daten oder Datenträger allgemein

(a)

## **Werkzeuge zur Erstellung von Flussdiagrammen**

Flussdiagramme können auf unterschiedliche Weise erstellt werden:

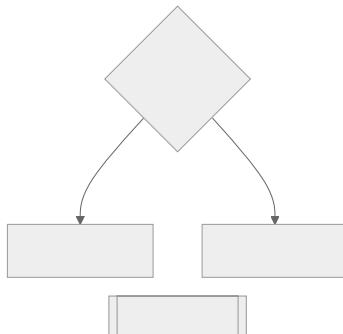
- mit Stift und Papier,
- mit Bürossoftwarepaketen wie LibreOffice,



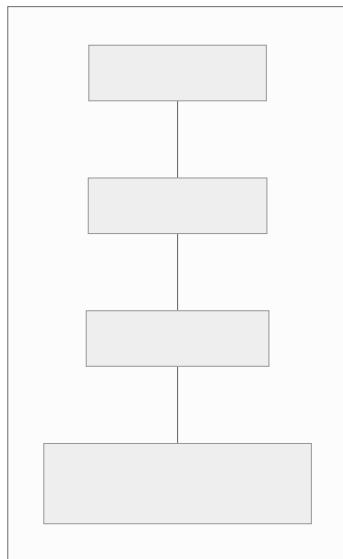
**Rechteck mit gerundeten Seiten (Nummer 6.4.1)** Grenzstelle zur Umwelt, die Beginn und Ende einer Folge anzeigen und beispielsweise Herkunft oder Verbleib von Daten signalisiert.



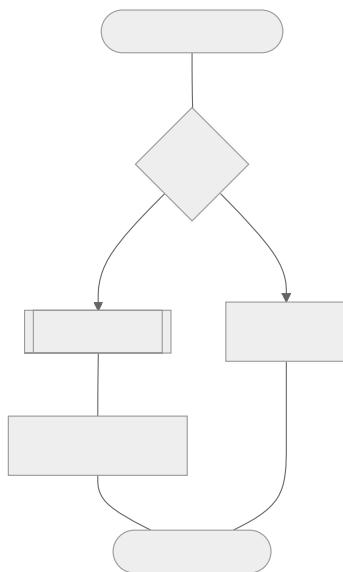
**Rechteck (Nummer 6.1.1)** Verarbeitung einschließlich Ein-/Ausgabe



**Raute (Nummer 6.1.2)** Verzweigung/Entscheidung



**Rechteck mit doppelten, vertikalen Linien (Nummer 7.2.4)**  
Hinweis auf ein an anderer Stelle dokumentiertes Programm



**Verbindung (Nummer 6.3.1)** Prozessketten werden mit Linien oder Pfeilen verbunden. Die Verbindungen sind von links nach rechts bzw. von oben nach unten orientiert, Abweichungen davon müssen mit Pfeilspitzen gekennzeichnet werden.

Sind mehrere Ausgänge mit Bedingungen verknüpft, müssen diese durch Beschriftung an den Verbindungslienken kenntlich gemacht werden.

Mehrere Verbindungen zu einem Sinnbild können zu einer Verbindung zusammengeführt werden. Sich kreuzende Verbindungslienken sollten aber vermieden werden. Diese stellen keine Zusammenführung dar. (*Hinweis: Für die Darstellung zusammengeföhrter Verbindungen wird Schemadraw empfohlen.*)

- Visualisierungsprogrammen wie [Graphviz](#) oder [Mermaid](#),
- mit spezialisierten Paketen wie [DiagrammeR](#) oder [Schemdraw](#).

Der Funktionsumfang und die Syntax unterscheiden sich und jedes Werkzeug hat eigene Stärken. Mermaid, Graphviz und Schemdraw sind deklarative Zeichenwerkzeuge, mit denen Flussdiagramme (und andere Grafiken) geschrieben und graphisch umgesetzt werden können. Mermaid hat eine einfache, intuitive Syntax. Graphviz erlaubt mehr Gestaltungsmöglichkeiten (benötigt in Python aber eine lokale Systeminstallation). Das Python-Modul Schemdraw unterstützt eine an Pythoncode angelehnte Syntax und bietet einen einfachen Zugriff auf DIN-konforme Knoten und Kanten.

**i** Note ???: Flussdiagramme in Python und R erstellen

## 5.11 Codebeispiel Python

```

# Flussdiagramm mit Schemdraw und Graphviz in Python

## Schemdraw
import subprocess
subprocess.call(['pip', 'install', 'schemdraw'])
import schemdraw

from schemdraw.flow import *

with schemdraw.Drawing() as d:
    # durchgehender Prozess in korrekter Reihenfolge von Start bis Ende
    d+= Start().label("Mittagspause")
    d+= Line().down()
    d+= Box().label("Kartoffeln abzählen")
    d+= Line().down()
    d+= (mit_Schale := Decision(S = "Ja", E = "Nein").label("Mit Schale?"))
    d+= Arrow().at(mit_Schale.S)
    d+= (kochen := Box().label("Kartoffeln kochen"))
    d+= Line().down()
    d+= Box().label("Kartoffeln essen")
    d+= Line().down()
    d+= Start().label("Pause Ende")

    # alternativer Ast der Verzweigung
    d+= Arrow().right().at(mit_Schale.E)
    d+= (schälen := Box().label("Kartoffeln schälen"))
    d+= Wire("|-", arrow = "->").at(schälen.S).to(kochen.E) # .S + .E (oder andere Richtung)

    d.draw()
    # d.save("Kartoffeln.svg")

## Graphviz
## benötigt eine lokale Installation von Graphviz https://graphviz.org/download/

import subprocess
subprocess.call(['pip', 'install', 'graphviz']) # installiert das Modul graphviz
import graphviz

dot = graphviz.Digraph(name = "Kartoffeln")

### Start / Ende
dot.node(name = "Mittagspause", shape = "Mrecord") # optional label
dot.node(name = "Pause Ende", shape = "Mrecord")

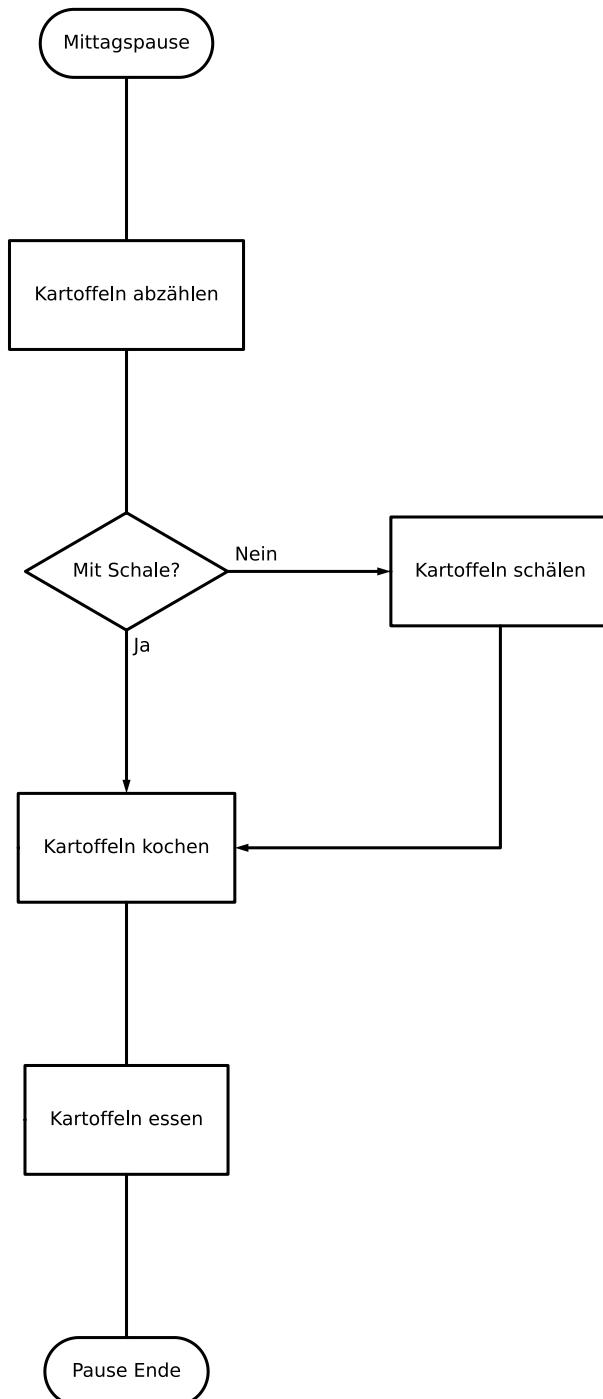
### Anweisungen
dot.node(name = "Kartoffeln abzählen", shape = "box")
dot.node(name = "Kartoffeln schälen", shape = "box")
dot.node(name = "Kartoffeln kochen", shape = "box")
dot.node(name = "Kartoffeln essen", shape = "box")

### Entscheidung
dot.node(name = "mit Schale", shape = "diamond")

```



## 5.12 Schemdraw in Python



*Hinweis: Aus technischen Gründen wurde die Grafik als Bild eingebunden.*

## 5.13 Codebeispiel R

```
# Flussdiagramm mit DiagrammeR in R

install.packages("DiagrammeR")
library("DiagrammeR")

## Mermaid
DiagrammeR::mermaid("
graph TD
    O(Mittagspause)
    A[Kartoffeln abzählen]
    B{mit Schale}
    D[Kartoffeln kochen]
    E[Kartoffeln essen]
    S[Kartoffeln schälen]
    Z(Pause Ende)

    O---A
    A---B
    B-->|ja|D
    B-->|nein|S
    S---D
    D---E
    E---Z
")

## Graphviz
DiagrammeR::grViz("

graph Kartoffeln {

    # defining nodes
    node [shape = Mrecord]

    Mittagspause; Pause_Ende

    node [shape = box]
```

```
Kartoffeln_abzählen
Kartoffeln_schälen
Kartoffeln_kochen
Kartoffeln_essen

node[shape = diamond]

mit_Schale

# defining edges
Mittagspause -- Kartoffeln_abzählen
Kartoffeln_abzählen -- mit_Schale
mit_Schale -- Kartoffeln_kochen[dir = forward label=ja]
mit_Schale -- Kartoffeln_schälen[dir = forward label=nein]
Kartoffeln_schälen -- Kartoffeln_kochen
Kartoffeln_kochen -- Kartoffeln_essen
Kartoffeln_essen -- Pause_Ende
}
")
```

## 5.14 Mermaid in R

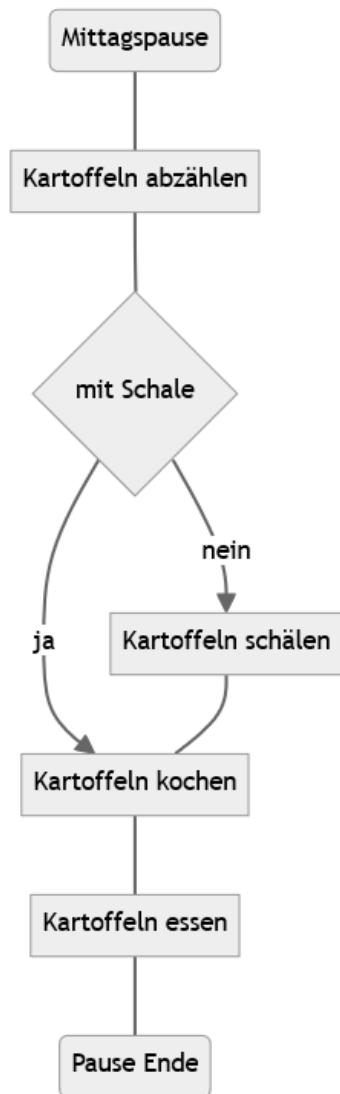


Figure 5.5:

## 5.15 Übungsaufgabe zählen

Erstellen Sie ein Flussdiagramm für ein Programm, das von 1-10 und von 15 bis 20 zählt.

## 5.16 Musterlösung Pseudocode

```
i = 0
SOLANGE i kleiner als 10 TUE
    erhöhe i um 1
    Ausgabe i
i = 14
SOLANGE i kleiner als 20 TUE
    erhöhe i um 1
    Ausgabe i
```

## 5.17 Musterlösung Mermaid

```
---
title: Zähle von 1-10 und von 15-20
---

flowchart TD
Start([Start])
initialisieren1[i = 0]
initialisieren2[i = 14]
Schleife1{i kleiner 10}
Schleife2{i kleiner 20}
Erhöhe1[i = i + 1]
Erhöhe2[i = i + 1]
Ausgabe1[Gebe i aus]
Ausgabe2[Gebe i aus]
Ende([Ende])

Start --- initialisieren1
initialisieren1 --- Schleife1

Schleife1 -->|nein| initialisieren2 --- Schleife2
Schleife2 -->|ja| Erhöhe2 --- Ausgabe2 --> Schleife2
Schleife2 -->|nein| Ende
```

```
Schleife1 -->|ja| Erhöhe1 --- Ausgabe1 --> Schleife1
```

```
%% unsichtbare Verbindungen zur Positionssteuerung
```

```
Ausgabe1 ~~~ Schleife2
```

```
Ausgabe2 ~~~ Ende
```

## 5.18 Musterlösung Flussdiagramm

Zähle von 1-10 und von 15-20

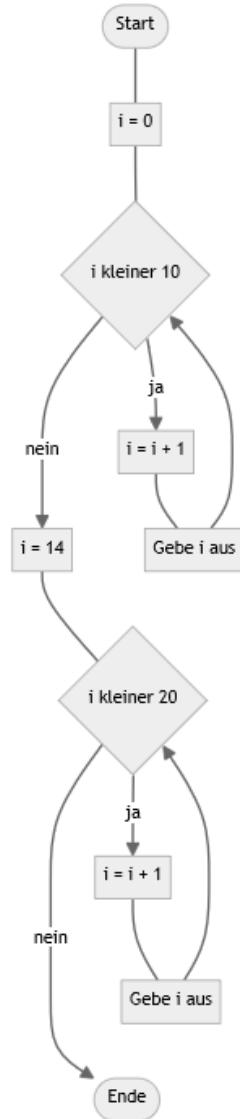


Figure 5.6: Flussdiagramm mit DiagrammeR in R

# 6 Musterlösungen

## 💡 Tip ???: Hefezopf backen

Für die Zubereitung des Hefezopfes werden mindestens drei Stunden benötigt. Der Teig (bzw. die Hefemischung) muss verteilt über mehrere Schritte insgesamt 120 Minuten gehen und nach dem Backen auskühlen. Aufgrund der langen Wartezeiten können die Arbeitsschritte von einer Person durchgeführt werden, eine parallele Bearbeitung durch eine zweite Person spart nur wenig Zeit ein. Lisa darf natürlich trotzdem eine Stunde früher vorbei kommen, aber dann ruht der Teig zum letzten Mal, bäckt im Ofen oder kühlt bereits aus. Zutaten mitbringen muss sie nicht, die Zutatenliste ist vollständig.

```
# Zwischenschritt 1 - ca. 25 Minuten
## Eingabe: 250 ml Milch, 475 g Weizenmehl, 1 Prise Zucker, ½ Würfel Hefe
## Werkzeug: Schüssel, Sieb, Topf, Löffel, Geschirrtuch
## Verarbeitung: Hefe in gezuckerter Milch lösen
## Ausgabe: Hefemischung, Mehl

BereiteHefemischung
  MehlSieben # benutze Sieb
  MehlHinzufügen
  MuldeBilden # Mulde im Mehl bilden
  HefeZerbröseln
  HefeHinzufügen
  MilchVorbereiten # benutze Topf
    **Solange** Milch < lauwarm **Tue**
    MilchErwärmen
  ZuckerHinzufügen
  Verrühren # Milch und Zucker mit Löffel vermischen
  MilchHinzufügen
  Verrühren # Hefe und Milchmischung mit Löffel vermischen
  TeigGeht(Zeit = 15 Minuten)
  SchüsselAbdecken # benutze Geschirrtuch
  WarmStellen

# Zwischenschritt 2 - ca. 75 Minuten
```

```

## Eingabe: Hefemischung, Mehl, 1 Ei, 60 g Zucker, 1 Prise Salz, 50 g Butter
## Werkzeug: Schüssel, Rührgerät, Messer, Geschirrtuch
## Verarbeitung: Ei, Milch, Salz, Butter hinzugeben und verkneten
## Ausgabe: Rohteig

BereiteTeig
  ButterWürfeln # benutze Messer
    # Ausgabe: n Butterwürfel
    **SOLANGE** TeigGeht **TUE**
      Warten
    EiHinzufügen
    MilchHinzufügen
    ZuckerHinzufügen
    SalzHinzufügen
    Kneten(Zeit = 3 Min, Stufe = niedrig) # benutze Rührhaken
    Kneten(Zeit = 5 Min, Stufe = hoch) # benutze Rührhaken
    ButterHinzufügen
    **SOLANGE** n > 0 **TUE**
      Kneten(Stufe = hoch)
      ButterwürfelHinzufügen
      n = n - 1
      Kneten(Zeit = 5 Min, Stufe = hoch) # benutze Rührhaken
    TeigGeht(Zeit = 60 Minuten)
    SchüsselAbdecken # benutze Geschirrtuch
    WarmStellen

# Zwischenschritt 3 - ca. 55 Minuten
## Eingabe: Rohteig, etwas Mehl
## Verarbeitung: Rohteig zu Teigzopf verarbeiten
## Werkzeug: Blech, Backpapier, Geschirrtuch
## Ausgabe: Teigzopf

ZopfFormen
  BackpapierAufBlechLegen
  ArbeitsflächeBemehlen
  TeigAufArbeitsflächeLegen
  TeigTeilen(Stücke = 3)
    # Ausgabe = n Teigstücke
  TeigRollen
  **SOLANGE** n > 0 **TUE**
    TeigstückRollen(Länge = 40 cm)

```

```

n = n - 1
ZopfErzeugen
  RollenFlechten
  EndenVerdrehen
ZopfAufBackpapierLegen
TeigGeht(Zeit = 45 Minuten)
  ZopfAbdecken # benutze Geschirrtuch
  WarmStellen

# Zwischenschritt 4 - ca. 40 Minuten
## Eingabe: Teigzopf, etwas Milch, etwas Hagelzucker
## Werkzeug: Pinsel, Ofen
## Verarbeitung: Zopf bestreichen, bestreuen und backen
## Ausgabe: Hefezopf (gebacken)

ZopfBacken
  OfenVorheizen
    **WENN** UmluftVerfügbar **DANN**
      Modus = Umluft
      Temperatur = 180 Grad
    **SONST**
      Modus = Ober- / Unterhitze
      Temperatur = 200 Grad
  ZopfMitMilchBestreichen # benutze Pinsel
  ZopfMitHagelzuckerBestreuen
  ZopfBacken
    BlechInOfenStellen
    **SOLANGE** Zopf < leicht bräunlich **TUE**
      BlechImOfenLassen
      BlechHerausholen
  ZopfAuskühlen
    **SOLANGE** Zopf > Zimmertemperatur **TUE**
      Warten

```

 Tip ???: Vitamin C bei Meerschweinchen

```

# Schritt 1 - Datensatz einlesen
## Eingabe: URL
## Werkzeug: Browser
## Verarbeitung: Speichern der Rohdaten
## Ausgabe: kommaseparierte Datei ToothGrowth.csv

```

```

HoleDatensatzVonURL
DatensatzSpeichern

# Schritt 2 - Mittelwertvergleich
## Eingabe: ToothGrowth.csv
## Werkzeug: IDE
## Verarbeitung: Mittelwerte für Teildatensätze bilden
## Ausgabe: Tabelle der Mittelwerte nach Dosis und Verabreichungsmethode

ZeilennummernBestimmen
Zeilennummern(supp = OJ)
Zeilennummern(supp = VC)
Zeilennummern(dose = 0.5)
Zeilennummern(dose = 1)
Zeilennummern(dose = 2)

TeildatensätzeBilden # über Kriterium oder über Zeilennummern
TeildatensatzSupp = OJ
    TeildatensatzDose = 0.5
    TeildatensatzDose = 1
    TeildatensatzDose = 2
TeildatensatzSupp = VC
    TeildatensatzDose = 0.5
    TeildatensatzDose = 1
    TeildatensatzDose = 2

MittelwerteErmitteln
MittelwertLen(Datensatz)
MittelwerteTeildatensätze
    MittelwertLen(TeildatensatzSupp = OJ & Dose = 0.5)
    MittelwertLen(TeildatensatzSupp = OJ & Dose = 1)
    MittelwertLen(TeildatensatzSupp = OJ & Dose = 2)
    MittelwertLen(TeildatensatzSupp = VC & Dose = 0.5)
    MittelwertLen(TeildatensatzSupp = VC & Dose = 1)
    MittelwertLen(TeildatensatzSupp = VC & Dose = 2)

TabelleErstellen
Spalten = OJ, VC
Zeilen = Dosis
Zellen = MittelwerteLen

TabelleAusgeben

# Schritt 3 - graphische Darstellung
## Eingabe: ToothGrowth.csv

```

```
## Werkzeug: Funktion für Boxplot
## Verarbeitung: Boxplot nach Dosis und Methode erzeugen
## Ausgabe: Boxplots nach Dosis und Verabreichungsmethode

Boxplot(Len nach Dosis & Methode)
  BoxplotErstellen
  VerabreichungsmethodeFarblichUnterscheiden
  LegendeEintragen
SpeichereBoxplot
```

## **7 Das Wichtigste**

In welcher Form und wie detailliert Sie Pseudocode formulieren, hängt von Ihrem Kenntnisstand und Ihrem Ziel ab. Die Entwicklung von Pseudocode kann als Kreativitätstechnik ohne formale Vorgaben eingesetzt werden, in detaillierter Form bei der Ausarbeitung einer algorithmischen Lösung helfen oder visuell in Form eines Flussdiagramms Ihre Kommunikation mit Dritten unterstützen.

# 8 Lernzielkontrolle

## 8.1 Kompetenzquiz

(1) Schreiben Sie ein Programm in Pseudocode, das alle geraden Zahlen bis 10 ausgibt.

(2) Fehler im Pseudocode finden.

Ein:e Freund:in zeigt Ihnen den Pseudocode eines Programms, das prüfen soll, ob Buchstaben groß oder klein geschrieben sind. Die Ausgabe klappt aber nur für klein geschriebene Buchstaben. Finden Sie den Fehler?

```
text = Eingabe("Bitte geben Sie Ihren Text ein.")
zeichen_liste = TeileZeichenweiseInListeAuf(text)

FÜR JEDES element IN zeichen_liste TUE

    Wenn Dezimalwert(element) >= 97 UND Dezimalwert(element) <= 122 TUE
        Schreibe("Das Zeichen ", element, "ist klein geschrieben.")

    Wenn Dezimalwert(element) >= 65 UND Dezimalwert(element) <= 90 TUE
        Schreibe("Das Zeichen ", element, "ist groß geschrieben.")
```

(3) Zeichnen Sie ein Flussdiagramm des korrekten Programms.

 Tip ???: Lösungen

Lösung 1:

```
i = 1
SOLANGE i kleiner gleich 10
    WENN i Modulo 2 gleich 0 TUE
```

Ausgabe i  
Erhöhe i um 1

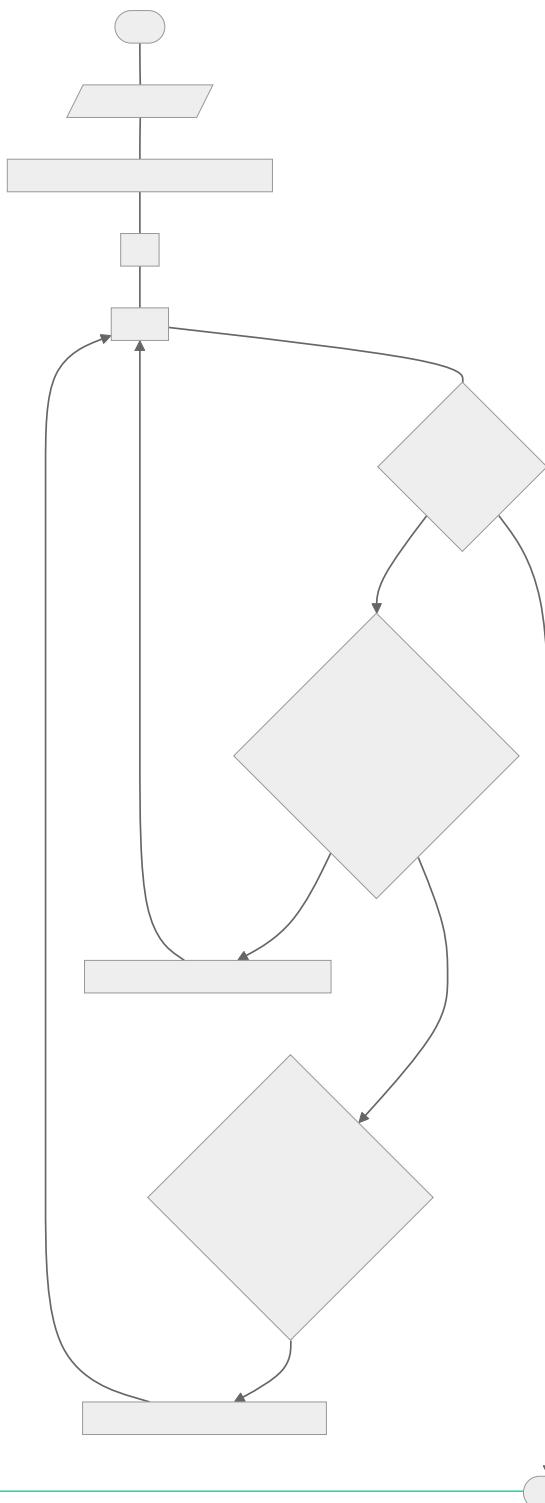
Lösung 2: Die Prüfung der Bedingungen für groß geschriebenen Buchstaben (Dezimalwert(element) >= 65 UND Dezimalwert(element) <= 90) erfolgt im Anweisungsblock, wenn ein Buchstabe klein geschrieben ist (Dezimalwert(element) >= 97 UND Dezimalwert(element) <= 122). Der Wahrheitswert dieser Bedingung ist falsch, wenn ein Buchstabe groß geschrieben ist. Die folgenden Anweisungen werden deshalb nicht ausgeführt, wenn element einen Großbuchstaben enthält. Durch korrektes Einrücken kann das Problem gelöst werden.

```
text = Eingabe("Bitte geben Sie Ihren Text ein.")  
zeichen_liste = TeileZeichenweiseInListeAuf(text)  
  
FÜR JEDES element IN zeichen_liste TUE  
  
    Wenn Dezimalwert(element) >= 97 UND Dezimalwert(element) <= 122 TUE  
        Schreibe("Das Zeichen ", element, "ist klein geschrieben.")  
  
    # korrigierte Einrückung  
    Wenn Dezimalwert(element) >= 65 UND Dezimalwert(element) <= 90 TUE  
        Schreibe("Das Zeichen ", element, "ist groß geschrieben.")
```

Lösung 3:

**Tip 8.1: Mermaid Diagramm**

prüfe Groß- / Kleinschreibung



## 8.2 Übungen

### 8.2.1 Börsenstrategie entwickeln

- 
- 

[Webseite](#)

[Direktlink zur XLS-Datei](#)

### **⚠ Warning ??: Datensatzbeschreibung S&P500**

Der Datensatz liegt in monatlicher Auflösung (**Date**) vor. Neben dem Kurs (**Price**) sind die ausgeschütteten Dividenden (**Dividend**) festgehalten.

*Hinweis: Der Datensatz liegt als Exceldatei vor und wurde aus Gründen der Übersichtlichkeit leicht bearbeitet (Anpassung der Spaltennamen, Formatierung Spalte Date, Runden der Spalten Price und Dividend). Bitte beachten Sie, dass wenn Sie den Datensatz selbst abrufen, das Erscheinungsbild entsprechend abweicht.*

#	Date	Price	Dividend
1	1871.01	4.44	0.26
2	1871.02	4.50	0.26
3	1871.03	4.61	0.26
4	1871.04	4.74	0.26
5	1871.05	4.86	0.26
6	1871.06	4.82	0.26
7	1871.07	4.73	0.26
8	1871.08	4.79	0.26
9	1871.09	4.84	0.26
10	1871.10	4.59	0.26
11	1871.11	4.64	0.26
12	1871.12	4.74	0.26
1828	2023.04	4121.47	68.38
1829	2023.05	4146.17	68.54
1830	2023.06	4345.37	68.71
1831	2023.07	4508.08	68.91
1832	2023.08	4457.36	69.11
1833	2023.09	4409.09	69.31
1834	2023.10	4269.40	69.64
1835	2023.11	4460.06	69.97
1836	2023.12	4685.05	70.30
1837	2024.01	4815.61	70.48
1838	2024.02	5011.96	70.65
1839	2024.03	5170.57	70.82

## **Part II**

# **w-python-minimal**

# Werkzeugbaustein Python

Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch und Sebastian Seipel. Werkzeugbaustein Python von Marc Fehr und Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar auf [GitHub](#). Ausgenommen von der Lizenz sind alle Logos Dritter und anders gekennzeichneten Inhalte. 2025



<https://github.com/bausteine-der-datenanalyse/w-python>

## Voraussetzungen

## Lernziele

- 
-

•

•

# **9 Einführung**

# 10 Willkommen bei Python!



Figure 10.1: Logo der Programmiersprache Python

GPLv3  
<https://www.python.org/psf/trademarks/>  
wikimedia

## 10.1 Lernziele dieses Kapitels

- 
- 
-

## 10.2 Ihr erstes Programm

```
print("Hallo Welt!")
```

## 10.3 Variablen – Namen für Werte

```
name = "Frau Müller"  
alter = 32
```

```
print(name + " ist " + str(alter) + " Jahre alt.")
```

 Aufgabe: Begrüßung mit Alter

Schreiben Sie ein Programm, das Sie mit Ihrem Namen begrüßt:

Hallo Frau Müller!

Tipp: In Python können Sie Texte mit `+` zusammenfügen. Denken Sie daran, dass Strings in Anführungszeichen stehen müssen.

**Lösung**

```
mein_name = "Ihr Name hier"  
print("Hallo " + mein_name + "!")
```

Hallo Ihr Name hier!

Erweitern Sie Ihr Programm so, dass es eine Begrüßung inklusive Alter ausgibt:

Hallo Frau Müller!  
Sie sind 32 Jahre alt.

Tipp: Verwenden Sie `print()` mehrmals oder fügen Sie Texte zusammen.

### Lösung

```
name = "Frau Müller"  
alter = 32  
  
print("Hallo " + name + "!")  
print("Sie sind " + str(alter) + " Jahre alt.")
```

Hallo Frau Müller!  
Sie sind 32 Jahre alt.

# **11 Datentypen verstehen**

## **11.1 Lernziele dieses Kapitels**

- 
- 
- 

## **11.2 Einleitung**

- 
- 

## **11.3 Die wichtigsten Datentypen**

```
wert = 42
print(type(wert)) # Ausgabe: <class 'int'>
```

## 11.4 Unterschiede zwischen int und float

•  
•

```
a = 10      # int
b = 2.5     # float

print("a:", a, "| Typ:", type(a))
print("b:", b, "| Typ:", type(b))
```

### Important

Die Unterscheidung ist wichtig: Manche Rechenoperationen verhalten sich je nach Datentyp leicht unterschiedlich.

## 11.5 Was sind Booleans (bool)?

•  
•  
  
•  
•  
•

```
ist_sonnig = True
hat_regenschirm = False

print("Sonnig:", ist_sonnig)
print("Regenschirm dabei?", hat_regenschirm)
print("Typ von 'ist_sonnig':", type(ist_sonnig))
```

## 11.6 Rechnen mit Zahlen

```
a = 10
b = 3

print("Addition:", a + b)
print("Subtraktion:", a - b)
print("Multiplikation:", a * b)
print("Potenzieren", a**b)
print("Division:", a / b)
print("Ganzzahlige Division:", a // b)
print("Division mit Rest:", a % b)
```

### **i** Note

```
// bedeutet: Ganzzahldivision, das Ergebnis wird abgerundet. Alternativ gibt es auch %.  
Hier wird eine Ganzzahldivision durchgeführt und der Rest ausgegeben.
```

## 11.7 Arbeiten mit Text

```
vorname = "Anna"  
nachname = "Beispiel"  
print("Willkommen, " + vorname + " " + nachname + "!")
```

```
punkte = 95  
print("Sie haben " + str(punkte) + " Punkte erreicht.")
```

## 11.8 Umwandlung von Datentypen (Typecasting)

```
# Beispiel: Zahl als Text anzeigen
punkte = 100
print("Sie haben " + str(punkte) + " Punkte.")

# Beispiel: String in Zahl umwandeln und berechnen
eingabe = "3.5"
wert = float(eingabe) * 2
print("Doppelt so viel:", wert)
```

#### 💡 Aufgabe: Alter in Tagen

Berechnen Sie, wie alt eine Person in Tagen ist.

```
alter_jahre = 32
tage = alter_jahre * 365
print("Sie sind ungefähr " + str(tage) + " Tage alt.")
```

Sie sind ungefähr 11680 Tage alt.

Tipp: Denken Sie an die Umwandlung in einen String, wenn Sie die Zahl ausgeben möchten.

#### Lösung

```
alter = 32
tage = alter * 365
print("Sie sind ungefähr " + str(tage) + " Tage alt.")
```

Sie sind ungefähr 11680 Tage alt.

# 12 Entscheidungen und Wiederholungen

•  
•

## 12.1 Lernziele dieses Kapitels

•  
•  
•

## 12.2 Bedingungen mit if, elif, else

```
alter = 17

if alter >= 18:
    print("Sie sind volljährig.")
else:
    print("Sie sind minderjährig.")
```

```
note = 2.3

if note <= 1.5:
    print("Sehr gut")
elif note <= 2.5:
    print("Gut")
elif note <= 3.5:
    print("Befriedigend")
else:
    print("Ausreichend oder schlechter")
```

## 12.3 Vergleichsoperatoren

## 12.4 Wiederholungen mit while

```
zähler = 0

while zähler < 5:
    print("Zähler ist:", zähler)
    zähler += 1
```

### Important

Achten Sie auf eine Abbruchbedingung – sonst läuft die Schleife endlos!  
Wird die Variable ‘zähler’ nicht erhöht, wird die Abbruchbedingung nie erreicht.

```
zähler = 0

while zähler < 5:
    print("Zähler ist:", zähler)
```

In Python können Sie die Programmausführung durch gemeinsames Drücken der Tasten Strg und C beenden. Je nach verwendeter Entwicklungsumgebung kann eine andere Tastenkombination gelten.

## 12.5 Schleifen mit for und range()

```
for i in range(5):
    print("Durchlauf:", i)
```

```
for i in range(1, 6):
    print(i)
```

## 12.6 Was macht range() genau?

### 12.6.1 Varianten:

```
range(5)
```

```
range(2, 6)
```

```
range(1, 10, 2)
```

💡 Aufgabe: Zähle von 1 bis 10

Nutzen Sie eine `for`-Schleife, um die Zahlen von 1 bis 10 auszugeben.

**Lösung**

```
for i in range(1, 11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

 Aufgabe: Gerade Zahlen ausgeben

Geben Sie alle geraden Zahlen von 0 bis 20 aus. Tipp: Eine Zahl ist gerade, wenn `zahl % 2 == 0`.

**Lösung**

```
for zahl in range(0, 21):
    if zahl % 2 == 0:
        print(zahl)
```

0  
2  
4  
6  
8  
10  
12  
14  
16  
18  
20

# 13 Mehrere Werte speichern

## 13.1 Was ist eine Liste?

```
namen = ["Ali", "Bente", "Carlos"]  
noten = [1.7, 2.3, 1.3, 2.0]
```

```
print(namen[0]) # erstes Element  
print(noten[-1]) # letztes Element
```

## 13.2 Teile aus Listen ausschneiden – Slicing

```
zahlen = [10, 20, 30, 40, 50, 60]  
print(zahlen[1:4]) # Ausgabe: [20, 30, 40]
```

### 13.2.1 Syntax: liste[start:stop]

- 
- 
- 
- 
- 

```
print(zahlen[:3])    # [10, 20, 30]
print(zahlen[3:])    # [40, 50, 60]
print(zahlen[:])     # vollständige Kopie
```

#### Note

Sie können auch mit negativen Indizes arbeiten (-1 ist das letzte Element):

```
print(zahlen[-3:])  # [40, 50, 60]
[40, 50, 60]
```

## 13.3 Über Listen iterieren

```
namen = ["Ali", "Bente", "Carlos"]

for name in namen:
    print("Hallo", name + "!")
```

## 13.4 Erweiterung: Bedingte Ausgaben

```
temperaturen = [14.2, 17.5, 19.0, 21.3, 18.4]

for t in temperaturen:
    if t > 18:
        print(t, "ist ein warmer Tag")
```

## 13.5 Durchschnitt berechnen

```
noten = [1.7, 2.3, 1.3, 2.0]

durchschnitt = sum(noten) / len(noten)
print("Durchschnittsnote:", round(durchschnitt, 2))
```

## 13.6 Listen erweitern: .append()

```
namen = []

namen.append("Ali")
namen.append("Bente")

print(namen)
```

### Note

Die Methode `.append()` hängt einen neuen Wert an das Ende der Liste.

## 13.7 Verschachtelte Schleifen

```
wochentage = ["Mo", "Di", "Mi"]
stunden = [1, 2, 3]

for tag in wochentage:
    for stunde in stunden:
        print(f"{tag}, Stunde {stunde}")
```

## 13.8 Listen sortieren

```
namen = ["Zoe", "Anna", "Ben"]
sortiert = sorted(namen)

print(sortiert)
```

### Important

Die Original-Liste bleibt **unverändert**.

Wenn Sie die Liste direkt verändern möchten, geht das mit:

```
namen.sort()
```

# 14 Wiederverwendbarer Code mit Funktionen

•  
•  
•

## 14.1 Lernziele dieses Kapitels

•  
•  
•

## 14.2 Eine Funktion definieren

- 1.
- 2.
- 3.
- 4.
- 5.

```
def hallo(name="Gast"):  
    begruessung = "Hallo " + name + "!"  
    return begruessung
```

```
def begruessung():
    print("Hallo und willkommen!")
```

```
begruessung()
```

## 14.3 Parameter übergeben

```
def begruessung(name):
    print("Hallo", name + "!")

begruessung("Alex")
```

## 14.4 Rückgabewerte mit return

```
def quadrat(zahl):
    return zahl * zahl

ergebnis = quadrat(5)
print(ergebnis)
```

## 14.5 Beispiel: Umrechnungen

### 14.5.1 Euro zu US-Dollar

```
def euro_zu_usd(betrag_euro):
    wechselkurs = 1.09
    return betrag_euro * wechselkurs

print("20 € sind", euro_zu_usd(20), "US-Dollar.")
```

#### 💡 Aufgabe: Begrüßung mit Name

Erstellen Sie eine Funktion `begruesse(name)`, die den Namen in einem Begrüßungstext verwendet:

```
Hallo Fatima, schön dich zu sehen!
```

#### Lösung

```
def begruesse(name):
    print("Hallo", name + ", schön dich zu sehen!")

begruesse("Fatima")
```

```
Hallo Fatima, schön dich zu sehen!
```

#### 💡 Aufgabe: Temperaturumrechnung

Schreiben Sie eine Funktion, die Celsius in Fahrenheit umrechnet:  
Formel: [  $F = C \times 1.8 + 32$  ]

#### Lösung

```
def celsius_zu_fahrenheit(c):
    return c * 1.8 + 32

print(celsius_zu_fahrenheit(20))
```

## 14.6 Parameter mit Standardwerten

```
def begruessung(name="Gast"):
    print("Hallo", name + "!")

begruessung()          # Hallo Gast!
begruessung("Maria")   # Hallo Maria!
```

### print() vs. return

Diese beiden Begriffe werden oft verwechselt:

	Ausdruck	Bedeutung
print()	zeigt einen Text auf dem Bildschirm	
return	gibt einen Wert an den Aufrufer zurück	

Beispiel:

```
def verdoppeln(x):
    return x * 2

# Ausgabe sichtbar machen
print(verdoppeln(5))  # Ausgabe: 10
```

10

# 15 Arbeiten mit Dateien

•  
•  
•

## 15.1 Lernziele dieses Kapitels

•  
•  
•  
•

## 15.2 Eine Datei einlesen

```
# Beispiel: Datei lesen
with open("01-daten/beispiel.txt", "r") as datei:
    inhalt = datei.read()
    print(inhalt)
```

•  
•  
•

## 15.3 Zeilenweise lesen

```
with open("01-daten/beispiel.txt", "r") as datei:  
    for zeile in datei:  
        print("Zeile:", zeile.strip())
```

### Note

.strip() entfernt Leerzeichen und Zeilenumbrüche am Anfang und Ende.

### Aufgabe: Datei lesen

Angenommen, es gibt eine Datei `gruesse.txt` mit folgendem Inhalt:

```
Hallo Anna  
Guten Morgen Ben  
Willkommen Carla
```

Schreiben Sie ein Programm, das jede Zeile einzeln einliest und mit `print(...)` wiedergibt.

### Lösung

```
with open("01-daten/gruesse.txt", "r") as f:  
    for zeile in f:  
        print(zeile.strip())
```

```
Hallo Anna  
Guten Morgen Ben  
Willkommen Carla
```

## 15.4 In eine Datei schreiben

```
with open("ausgabe.txt", "w") as datei:  
    datei.write("Das ist eine neue Zeile.\n")  
    datei.write("Und noch eine.")
```

•  
•

## 15.5 Zeilenweise schreiben mit Schleife

```
daten = ["Apfel", "Banane", "Kirsche"]

with open("obst.txt", "w") as f:
    for eintrag in daten:
        f.write(eintrag + "\n")
```

! Important

Jede Zeile endet mit \n für einen Zeilenumbruch.

💡 Aufgabe: Liste in Datei schreiben

Gegeben ist eine Liste von Städten:

```
staedte = ["Berlin", "Hamburg", "München"]
```

- Schreiben Sie ein Programm, das jede Stadt in eine neue Zeile einer Datei staedte.txt schreibt.

Lösung

```
staedte = ["Berlin", "Hamburg", "München"]

with open("staedte.txt", "w") as f:
    for stadt in staedte:
        f.write(stadt + "\n")
```

## 15.6 Alle Zeilen auf einmal lesen mit readlines()

```
with open("01-daten/beispiel.txt", "r") as f:  
    zeilen = f.readlines()  
    print(zeilen)
```

! Important

Jede Zeile endet mit \n, deshalb kann eine Nachbearbeitung mit .strip() sinnvoll sein:

```
for zeile in zeilen:  
    print(zeile.strip())
```

Dies ist ein Test.

## 15.7 Dateien manuell schließen mit close()

```
datei = open("01-daten/beispiel.txt", "w")  
datei.write("Dies ist ein Test.")  
datei.close()
```

! Important

close() ist wichtig, damit Änderungen gespeichert werden und die Datei nicht gesperrt bleibt.

**Empfehlung:** Nutzen Sie immer `with open(...)`, da Python die Datei dann automatisch schließt – auch bei Fehlern.

## **16 Das Wichtigste**

# **Part III**

## **w-python**

# Werkzeugbaustein Python



Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch und Sebastian Seipel. Werkzeugbaustein Python von Marc Fehr und Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar auf [GitHub](#). Ausgenommen von der Lizenz sind alle Logos Dritter und anders gekennzeichneten Inhalte. 2025

<https://github.com/bausteine-der-datenanalyse/w-python>

## Voraussetzungen

[Werkzeugbaustein Pseudocode](#)

## Lernziele

-

•

•

•

•

•

# 17 Einleitung: Datenanalyse mit Python



Figure 17.1: Logo der Programmiersprache Python

GPLv3  
<https://www.python.org/psf/trademarks/>  
wikimedia

UTF-8

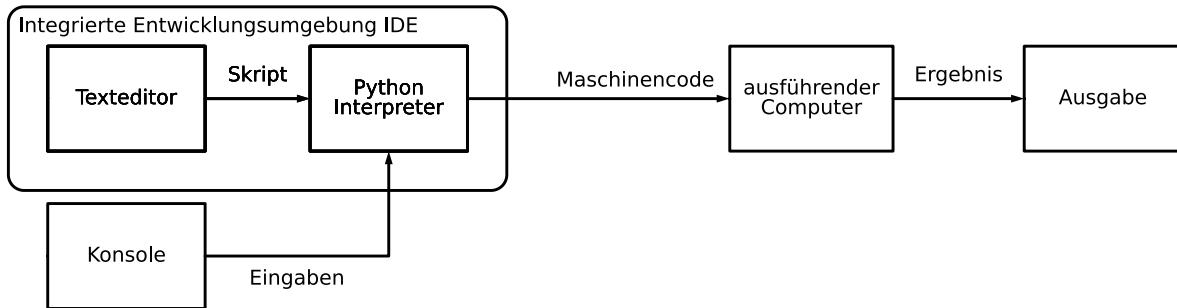


Figure 17.2: Programmentwicklung mit Python

## 17.1 Grundbegriffe der objektorientierten Programmierung

### 17.1.1 Klassen, Typen, Objekte, Attribute

```

print(type(2), 2 + 2, "Ganzzahlen werden addiert.")
print(type('a' and '2'), 'a' + '2', "Zeichen werden verkettet.")
print(type(True), True + True, "Wahrheitswerte werden addiert.")

```

```
print(None + None)
```

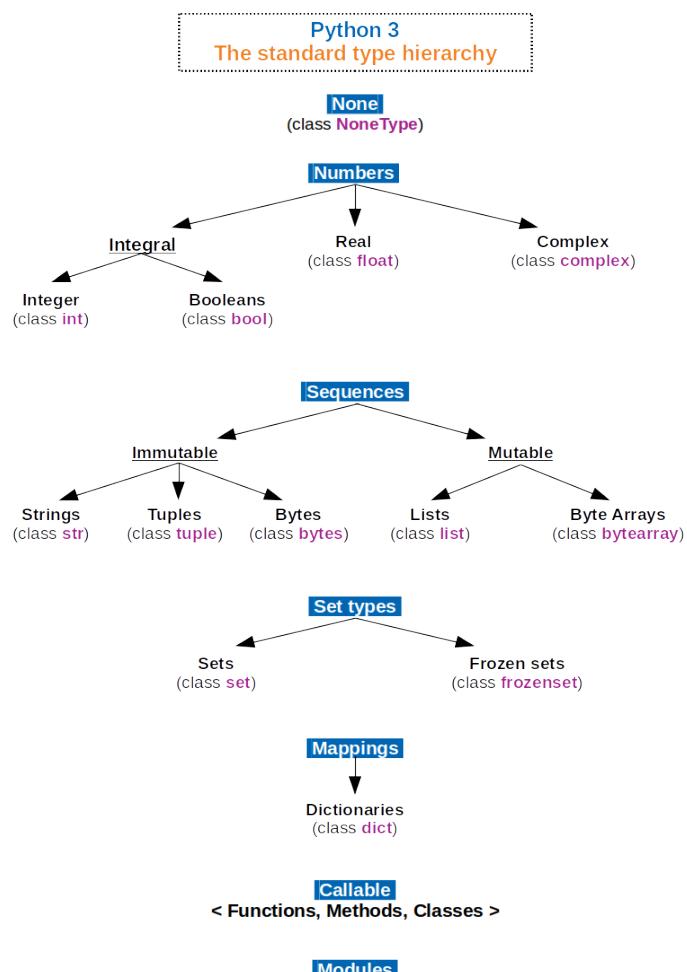


Figure 17.3: Datentypen in Python

```
print(type(print))
```

## 17.2 Programmcode formatieren

1.

```
print(1 + 2)
print('123: Hallo Welt!')
text_variable = 'Hallo Python!'
print(text_variable)
```

2.

```
# Ein reiner Kommentar
# print("Python ist großartig!") # auskommentierter Code, gefolgt von einem Kommentar
print("Python ist ziemlich gut.") # auszuführender Code, gefolgt von einem Kommentar
```

3.

```
variable1 = 15
variable2 = 25

# Zeilenfortsetzung mit \
summe = variable1 + \
        variable2

# Zeilenfortsetzung innerhalb einer Funktion
print(variable1,
      variable2,
      summe)
```

4.

```
print(1+0, 1 + 1, 1 + 2)
```

5.

```
for i in range(2):
    print(variable1)
    print(variable2)
print(summe)
```

## 17.3 Ausgabe formatieren

```
zahl1 = 5
zahl2 = 7
verhältnis = zahl1 / zahl2
print(f"Das Verhältnis von {zahl1} zu {zahl2} ist {verhältnis}.")
```

```
•  
•  
•  
•
```

```
print(f"Das Verhältnis von {zahl1} zu {zahl2} ist {verhältnis:.2f}.")
```

```
print(f"Das Verhältnis ist genauer {0.7142857142857143:.3f}.")
```

```
print(f"Das Verhältnis von {zahl1} zu {zahl2} ist {verhältnis:7.2f}.")  
print(f"Das Verhältnis ist genauer {0.7142857142857143:07.3f}.")
```

## **17.4 Ganze Zahlen**

## **17.5 Gleitkommazahlen**

## **17.6 Zeichenfolgen**

[Python Dokumentation](#)

## 17.7 Aufgaben

1.

•

•

2.

3.

💡 Tip ??: Musterlösung Ausgabe

### 1. Aufgabe

```
# Verändern Sie die natürliche Schreibweise so, dass nur noch eine Stelle nach dem Komma anfällt
# Was fällt auf?
print(f"natürliche Schreibweise: {1015.39:12.4f}")
print(f"wissenschaftliche Schreibweise: {1015.39:e}")

print(f"Schreibweise mit einer Kommastelle: {1015.39:12.1f}")
```

```
# Verändern Sie die wissenschaftliche Schreibweise so, dass anstelle von e die Zehnerbasis verändert wird
print(f"Zehnerbasis verändern: {1015.39:E}")
```

```
natürliche Schreibweise:      1015.3900
wissenschaftliche Schreibweise: 1.015390e+03
Schreibweise mit einer Kommastelle:      1015.4
Zehnerbasis verändern: 1.015390E+03
```

Es fällt auf, dass die Nachkommastelle der natürlichen Zahl automatisch gerundet wird. Beim Ändern der Zehnerbasis zu einem großen **E** wird dieses auch in der `print`-Ausgabe groß geschrieben.

### 2. Aufgabe

```
# 2.
# Wandeln Sie die Zahl 1015.39 in eine Zeichenfolge um und stellen Sie diese mit 12 Stellen dar
print(f"{"1015.39":>12s}")
```

1015.39

### 3. Aufgabe

```
# 3.  
# Geben Sie mit Hilfe der formatierten Zeichenfolge eine Tabelle aus, welche  
# die Spalten x, x2 und x3 und für ganze Zahlen zwischen -2 und 2 auflistet.  
  
x = -2  
print(f"{x:>10d} {x**2:>10d} {x**3:>10d}")  
x = -1  
print(f"{x:>10d} {x**2:>10d} {x**3:>10d}")  
x = 0  
print(f"{x:>10d} {x**2:>10d} {x**3:>10d}")  
x = 1  
print(f"{x:>10d} {x**2:>10d} {x**3:>10d}")  
x = 2  
print(f"{x:>10d} {x**2:>10d} {x**3:>10d}")
```

-2	4	-8
-1	1	-1
0	0	0
1	1	1
2	4	8

Alternativ ist auch eine (elegantere) Lösung mit einer `for`-Schleife möglich:

```
for x in range(-2,3):  
    print(f"{x:>12d} {x**2:>12d} {x**3:>12d}")
```

-2	4	-8
-1	1	-1
0	0	0
1	1	1
2	4	8

Musterlösung von Marc Sönnecken.

# 18 Datentypen

## 18.1 Zahlen

### 18.1.1 Ganzzahlen

```
print(12, -8)  
•  
print(0b1000, "plus", 0b0000, "plus", 0b0010, \  
      "plus", 0b0001, "ist", 0b1011)
```

```
print(0o7000, "plus", 0o0700, "plus", 0o0020, \
      "plus", 0o0000, "ist", 0o7720)
```

•

```
print(0xF000, "plus", 0x0200, "plus", 0x00A0, \
      "plus", 0x0001, "ist", 0xF2A1)
```

### 18.1.2 Gleitkommazahlen

```
print(120.6, 1206e-1, 12060e-2, "\n")

print("Beim Lottogewinn in Exponentialschreibweise zählt das Vorzeichen.")
print(1e-3, "oder", 1e+3)
```

```
print(0.1) # Die kürzeste Dezimalzahl zur Binärapproximation  
print(format(0.1, '.17g')) # Die nächstlängere Dezimalzahl zur selben Binärapproximation  
print(0.3 - 0.2) # binär gerechnet, ändert sich die Binärapproximation
```

```
print(0.1 + 0.2)  
print(0.01 + 0.02)  
print(0.001 + 0.002)  
print(0.0001 + 0.0002)  
print(0.00001 + 0.00002)
```

## 18.2 Arithmetische Operatoren

### 18.3 Aufgaben Zahlen

- 1.
- 2.
- 3.
- 4.
- 5.
- 6.
7.
  - a)
  - b)

💡 Tip ??: Musterlösung Zahlen

```
# 1.  
print('4 + 2 * 4 =', 4+2*4, '\n')  
# 2.  
print('2 hoch 12 =', 2**12, '\n')  
# 3.  
print('Rest aus 315 geteilt durch 4 =', 315%4, '\n')  
# 4.  
print('1 + 2^6 / 5 = ', 1+(2**6)/5 , '\n')  
# 5.  
print('binär 11111101001 ist dezimal =', 0b11111101001, '\n')  
# 6.  
print('binär 11111101001 / 101 ist dezimal =', 0b11111101001/0b101, '\n')  
# 7.  
variante_a = 1000*1.03**20  
print(f'nach 20 Jahren mit 3 Prozent = {variante_a:.2f}')  
variante_b = 1000*1.02**30  
print(f'nach 30 Jahren mit 2 Prozent = {variante_b:.2f}')
```

4 + 2 \* 4 = 12

2 hoch 12 = 4096

Rest aus 315 geteilt durch 4 = 3

1 + 2^6 / 5 = 13.8

binär 11111101001 ist dezimal = 2025

binär 11111101001 / 101 ist dezimal = 405.0

nach 20 Jahren mit 3 Prozent = 1806.11

nach 30 Jahren mit 2 Prozent = 1811.36

Musterlösung von Marc Sönnecken

## 18.4 Boolesche Werte

```
print("Ist 10 größer als 9?", 10 > 9)
print("Ist 11 kleiner als 10?", 11 < 10)
print("Ist 10 genau 10.0?", 10 == 10.0, "\n")

print("True und False können mit + addiert werden:", True + False)
print("... und mit * multipliziert werden:", True * False, "\n")
```

```
print("Ist 10 > 9 UND 10 > 8?", (10 > 9) * (10 > 8))
```

```
print("Ist 10 > 9 UND > 8?", bool((10 > 9) * (10 > 8)))
```

```
print(bool(1), bool(2), bool(2.4))
print(bool('a'), bool('b'), bool('ab'))
```

```
print(bool(False), bool(0))
print(bool("")) # eine leere Zeichenfolge
print(bool([])) # eine leere Liste
print(bool(())) # eine leeres Tupel
print(bool({})) # ein leeres Dictionary
print(bool(None)) # None deklariert einen nicht existenten Wert
```

```
meine_Liste = ['Äpfel', 'Butter']
if meine_Liste:
    print(f"Wir müssen {meine_Liste} einkaufen.")

meine_Liste = [] # eine leere Liste
if meine_Liste:
    print(f"Wir müssen {meine_Liste} einkaufen.")
```

## 18.5 Logische Operatoren

### **⚠ Warning ??: Bitweise Operatoren**

Besondere Vorsicht ist mit den **bitweisen Operatoren** geboten. Diese vergleichen Zahlen nicht als Ganzes, sondern stellenweise (im Binärsystem). Zu beachten ist, dass die bitweisen Operatoren Ausführungsriorität vor Vergleichsoperationen haben.

```
print(10 > 5 and 10 > 6)
print(10 > 5 & 10 > 6)
print(5 & 10)
print(10 > False > 6)
print((10 > 5) & (10 > 6))
```

```
True
False
0
False
True
```

#### **Tip**

Im Allgemeinen werden die bitweisen Operatoren für die Datenanalyse nicht benötigt. Vermeiden Sie unnötige Fehler: Vermeiden Sie die bitweisen Operatoren **&**, **^** und **|**.

Die Operatoren **&**, **^** und **|** haben jedoch für Mengen (die wir später kennenlernen werden) eine andere Bedeutung. Auch in anderen Modulen kommt den Operatoren syntaktisch eine andere Bedeutung zu, bspw. im Paket Pandas bei der Übergabe mehrerer Slicing-Bedingungen `df = df [(Bedingung1) & (Bedingung2) | (Bedingung3)]`.

## 18.6 Aufgaben boolsche Werte

- 1.
- 2.
- 3.
- 4.
- 5.

💡 Tip ???: Musterlösung boolsche Werte

```
# 1.  
print('Ist 44/4,5 größer als 10?', 44/4.5 > 10)  
# 2.  
print('Ist 4,5 größer als 4 aber kleiner als 5?', 4<4.5 and 4.5<5)  
# 3.  
print('Ist 2 hoch 10 gleich 1024?', 2**10 == 1024)  
# 4.  
print('Sind die Zahlen 3, 4 und 5 ganzzahlig durch 2 teilbar ODER ungleich 10?')  
print('3:', 3%2 == 0 or 3 != 10)  
print('4:', 4%2 == 0 or 4 != 10)  
print('5:', 5%2 == 0 or 5 != 10)  
# 5. Alter prüfen  
  
# Variable für das zu prüfende Alter  
age = 17  
print('Alter =', age)  
  
print('Die Person muss den Vollpreis zahlen:', age >= 18 and age < 65)  
print('Die Person fährt mit Rabatt:', age >= 14 and age < 18 or age >= 65)  
print('Die Person fährt kostenlos:', age < 14)
```

```
Ist 44/4,5 größer als 10? False  
Ist 4,5 größer als 4 aber kleiner als 5? True  
Ist 2 hoch 10 gleich 1024? True  
Sind die Zahlen 3, 4 und 5 ganzzahlig durch 2 teilbar ODER ungleich 10?  
3: True  
4: True
```

```
5: True  
Alter = 17  
Die Person muss den Vollpreis zahlen: False  
Die Person fährt mit Rabatt: True  
Die Person fährt kostenlos: False
```

Musterlösung von Marc Sönnecken und Maik Poetzsch

## 18.7 Zeichenfolgen

```
print('eine Zeichenfolge')  
print("noch eine Zeichenfolge")
```

```
print('A sophisticated heap beam, which we call a "LASER".')  
print("I've turned the moon into what I like to call a 'Death Star'.")
```

```
print("Das Steuerzeichen \\ ermöglicht die gleichen \"Anführungszeichen\" in der Ausgabe von  
print("Erst ein\tTabstopp, dann eine\nneue Zeile.")
```

```
print("Die Daten liegen unter: C:\tolle_daten\nordpol\weihnachtsmann")
print(r"Die Daten liegen unter: C:\tolle_daten\nordpol\weihnachtsmann")
```

## 18.8 Operationen mit Zeichenfolgen

```
# string + string
print('a' + 'b')

# string + Zahl
print(15 * 'a')

# logische Operatoren
print('a' < 'b', 'a' >= 'b', 'a' != 'b')
print('a' or 'b', 'a' and 'b')
```

## 18.9 Aufgaben Zeichenfolgen

- 1.
- 2.
- 3.
- 4.

💡 Tip ??: Musterlösung Aufgaben Zeichenfolgen

```
# 1.  
print("python" + "for beginners") # Die beiden Zeichenfolgen werden direkt aneinander gekettet  
# 2.  
print(10*("tick tack"))  
# 3.  
print( "Aachen" < "Bern" ) # es werden die Unicode-Werte verglichen  
# 4.  
print(r"~\home\tobi\neue_daten") # Python interpretiert den Backslash "\\" als Escape-Zeichen
```

```
pythonfor beginners  
tick tacktick tacktick tacktick tacktick tacktick tacktick tacktick tacktick tack  
True  
~\home\tobi\neue_daten
```

Musterlösung von Marc Sönnecken

## 18.10 Variablen

```
var_1 = 'ABC'  
var_2 = 26  
var_3 = True  
  
print("Das", var_1, "hat", var_2, "Buchstaben. Das ist", var_3)
```

```
print("Die Variable var_1 hat den Typ", type(var_1))  
print("Die Variable var_2 hat den Typ", type(var_2))  
print("Die Variable var_3 hat den Typ", type(var_3))
```

```
var_1 = 100
print("Die Variable var_1 hat den Typ", type(var_1))
```

```
var_1 = var_1 / 11
print("Die Variable var_1 hat den Typ", type(var_1))
```

```
a = 67
print(a, type(a))

b = a + 1.8
print(b, type(b), "\n")

print(f"Beachten Sie das Abschneiden der Nachkommastelle:\nUmwandlung in Ganzzahlen mit int()\n\nprint(f"Umwandlung in ASCII-Zeichen mit chr(): {( a := chr(a) ), ( b := chr(b) )}\n")
print(f"Umwandlung in eine ASCII-Zahl mit ord(): {( a := ord(a) ), ( b := ord(b) )}\n")
print(f"Umwandlung in Fließkommazahlen mit float(): {( a := float(a) ), ( b := float(b) )}\n")
```

```
print(f"Umwandlung in Zeichen mit str(): {( a := str(a) ), ( b:= str(b) )}\n")  
print(f"Umwandlung in Wahrheitswerte mit bool(): {bool(a), bool(b)}")
```

### 18.10.1 Weitere Zuweisungsoperatoren

Walross-Operator

[Python-Dokumentation](#)

[hier](#)

### 💡 Tip ??: Lesbare Zuweisungen

Die in der Tabelle gezeigten Zuweisungsoperatoren sind für jemanden, der\* die Ihren Code liest, gut zu lesen und nachzuvollziehen, da Zuweisungen immer am Beginn einer Zeile, also ganz links, stehen.

Dagegen kann der Walross-Operator an einer beliebigen Stelle in Ihrem Code stehen. Das mag für Sie beim Schreiben ein Vorteil sein, der Lesbarkeit ist das aber abträglich. Wenn Sie den Walross-Operator verwenden, achten Sie deshalb auf die Nachvollziehbarkeit Ihres Codes.

## 18.10.2 Benennung von Variablen

### 💡 Tip ??: Auflösung Variablen

```
print(var_3, type(var_3))
print(variable1, type(variable1))
print(a, type(a))
```

```
True <class 'bool'>
15 <class 'int'>
67.0 <class 'str'>
```

[diesem Wikipedia Abschnitt](#)

### Warning ??: Schlüsselwörter und Funktionsnamen

In Python reservierte Schlüsselwörter und Funktionsnamen sind ungeeignete Variablennamen. Während Python die Wertzuweisung zu Schlüsselwörtern wie `True` oder `break` mit einem Syntaxfehler quittiert, lassen sich Funktionsnamen neue Werte zuweisen, beispielsweise mit `print = 6`. Wenn Sie die Funktion `print` dann aufrufen, funktioniert diese natürlich nicht mehr. In diesem Fall müssen Sie die Zuweisung aus dem Skript entfernen und Python neu starten.

Folgende Schlüsselwörter gibt es in Python:

```
and      continue    for      lambda      try
as       def         from     nonlocal    while
assert   del         global   not        with
async    elif        if       or         yield
await    else        import   pass       True
break   except      in       raise      class
False   finally    is       return    None
```

## 18.11 Aufgaben Variablen

1.

a)

b)

c)

2.

💡 Tip ??: Musterlösung Aufgaben Variablen

1. Aufgabe

```
# 1.

def zeit_aufteilung(t):
    tage = t // (24*3600)
    t = t % (24*3600)
    stunden = t//3600
    t = t%3600
    Minuten = t//60
    sekunden = t%60

    return tage, stunden, Minuten, sekunden

# hier die zu prüfenden Werte a, b, c eintragen
zeit_gegeben = 79222

tage, stunden, Minuten, sekunden = zeit_aufteilung(zeit_gegeben)

print(f"{zeit_gegeben} Sekunden entsprechen {tage} Tagen, {stunden} Stunden, {Minuten} Minuten und {sekunden} Sekunden")
```

79222 Sekunden entsprechen 0 Tagen, 22 Stunden, 0 Minuten und 22 Sekunden

2. Aufgabe

```
# 2.

# Ausgangswerte inkl. Umrechnungen
x0 = 10*1000 # Kilometer in Metern
v0 = 50/3.6 # km/h in m/s
a = 0.1
t = 10*60 # Minuten in Sekunden

x = x0 + v0 * t + 0.5 * a * t **2

print(f"Die berechnete Position des Fahrzeugs zur Zeit t = {(t/60):.0f} Minuten beträgt {(x/1000):.0f} Kilometer")
```

Die berechnete Position des Fahrzeugs zur Zeit  $t = 10$  Minuten beträgt 36.33 Kilometer.

Musterlösung von Marc Sönnecken

# 19 Funktionen: Grundlagen

[Dokumentation](#)

## 💡 Tip ??: Dokumentation

Der wichtigste Tipp zuerst: **Benutzen Sie die Dokumentation!** Auch wenn Sie eine Funktion kennen: Vergewissern Sie sich regelmäßig, dass Sie noch auf dem neuesten Stand sind. Auf diese Weise erhalten Sie einen vollständigen Überblick über standardmäßig gesetzte und optional verfügbare Parameter. Außerdem erkennen Sie Änderungen in der Programmausführung und vermeiden so unerwartete Fehler.

 **Added in version 1.5.0:** Support for `defaultdict` was added.  
Specify a `defaultdict` as input where the default determines the `dtype` of the columns which are not explicitly listed.

(a) Neuerung in Python

 **Deprecated since version 2.0.0:** A strict version of this argument is now the default, passing it has no effect.

(a) Abkündigung in Python

Achten Sie auf die korrekte Version der Dokumentation.

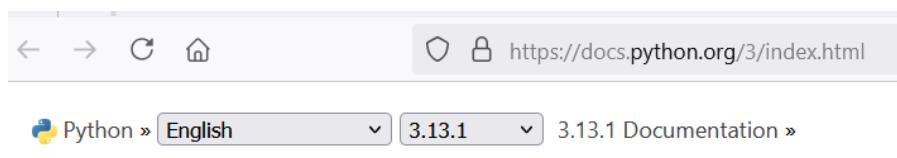


Figure 19.3: Versionsauswahl der Dokumentation

## 19.1 Funktionen und Methoden

### 19.1.1 Funktionen

```
var_str = 'ABC'  
var_int = 26  
var_bool = True  
  
print("Die Variable var_1 hat den Typ", type(var_str))  
print("Die Variable var_2 hat den Typ", type(var_int))  
print("Die Variable var_3 hat den Typ", type(var_bool))
```

```
res = print( 15 )  
print(res)
```

```
print(bool(sum([1, 2])))
```

## 19.1.2 Methoden

```
toller_text = "Python 3.12 ist großartig."  
  
print(toller_text.upper())  
print(toller_text.lower())  
print(toller_text.title(), "\n")  
  
print(("Mit in Klammern gesetzten Werten klappt es auch.").upper())
```

```
print((1).upper())
```

```
print('Katze'.lower().count('k'))
```

**i** Note ???: Methoden eines Objekts bestimmen

Mit der Funktion `dir(Objekt)` können die verfügbaren Methoden eines Objekts ausgegeben werden. Dabei werden jedoch auch die Attribute und die Methoden der Klasse des Objekts ausgegeben, sodass die Ausgabe oft sehr umfangreich ist. Zum Beispiel für die Ganzzahl 1:

```
print(dir(1))
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__di...
```

Um die Ausgabe auf Methoden einzuschränken, kann folgende Funktion in Listenschreibweise verwendet werden:

```
objekt = 1
```

```
attribute = [attr for attr in dir(objekt) if callable(getattr(objekt, attr))]
print(attribute)
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__di...
```

Mit doppelten Unterstrichen umschlossene Methoden sind für die Klasse definierte Methoden. Folgende Funktion entfernt Methoden mit doppelten Unterstrichen aus der Ausgabe:

```
objekt = 1
```

```
attribute = [attr for attr in dir(objekt) if (callable(getattr(objekt, attr)) and not attr...
```

```
['as_integer_ratio', 'bit_count', 'bit_length', 'conjugate', 'from_bytes', 'is_integer', 't...
```

Im Fall einer Ganzzahl können Methoden (zur Abgrenzung von Gleitkommazahlen in umschließenden Klammern) wie folgt aufgerufen werden:

```
(1).as_integer_ratio()
```

```
(1, 1)
```

Die Methoden des Objekts 'toller\_text':

```
objekt = toller_text
```

```
attribute = [attr for attr in dir(objekt) if (callable(getattr(objekt, attr)) and not attr...
```

```
['capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', ...
```

## 19.2 Parameter

[Python-Dokumentation](#) [Parameter](#)

[Dokumentation der Funktion](#)

- 
- 

### 💡 Tip ???: Ausnahmen bei Standardwerten

Bei den in der Funktionsdefinition genannten Werten handelt es sich nicht immer um die tatsächlichen Standardwerte. Es empfiehlt sich deshalb, wenn eine Funktion verwendet wird, die Beschreibung der Parameter zu lesen.

Einige Funktionen verwenden das Schlüsselwort `None` zur Kennzeichnung des Standardwerts. Der Wert `None` dient dabei als Platzhalter. Ein Beispiel ist die NumPy-Funktion [numpy.loadtxt\(\)](#).

```
numpy.loadtxt(fname, dtype=<class 'float'>, comments='#', delimiter=None, converters=None,
              skiprows=0, usecols=None, unpack=False, ndmin=0, encoding=None, max_rows=None,
              *, quotechar=None, like=None)
```

- Für den Parameter `delimiter` ist als Standardwert das Schlüsselwort `None` einge tragen. Wie der Funktionsbeschreibung zu entnehmen ist, ist der Standartwert tatsächlich das Leerzeichen: “The default is whitespace.”
- Auch der Parameter `usecols` hat den Standarwert `None`: “The default, None, re sults in all columns being read.”

Ein weiteres Beispiel ist die Funktion [pandas.read\\_csv\(\)](#). Einige Argumente haben den Standardwert `<no_default>`. (Im Folgenden werden nur ausgewählte Parameter gezeigt).

```
pandas.read_csv(sep=<no_default>, verbose=<no_default>)
```

Aus der Beschreibung können die tatsächlichen Standardwerte abgelesen werden:  
sep : str, default ','  
verbose : bool, default False

- 

#### 💡 Tip ??: Positionale und Schlüsselwortargumente, \*args und \*\*kwargs

Die Symbole \* und / zeigen an, welche Parameter positional und welche per Schlüsselwort übergeben werden können bzw. müssen.

Linke Seite	Trennzeichen	Rechte Seite
nur positionale Argumente	/	positionale oder Schlüsselwortargumente
positionale oder Schlüsselwortargumente	*	nur Schlüsselwortargumente

(<https://realpython.com/python-asterisk-and-slash-special-parameters/>)

Ein Beispiel für das Trennzeichen \* ist die Funktion `glob` aus dem gleichnamigen Modul. Der Parameter `pathname` kann positional (an erster Stelle) oder als Schlüsselwort übergeben werden. Die übrigen Parameter müssen als Schlüsselwortargumente übergeben werden.

```
glob.glob(pathname, *, root_dir=None, dir_fd=None, recursive=False, include_hidden=False)
```

Beide Steuerzeichen können innerhalb einer Funktionsdefinition vorkommen, allerdings nur in der Reihenfolge / und \*. Im umgekehrten Fall wäre es unmöglich, Argumente zu übergeben. Ein Beispiel ist die Funktion `sorted`. Der erste Parameter muss positional übergeben werden, die Parameter `key` und `reverse` müssen als Schlüsselwörter übergeben werden.

```
sorted(iterable, /, *, key=None, reverse=False)
```

## Ausnahmen

Einige Funktionen weichen von der Systematik ab, beispielsweise die Funktionen `min()` und `max()`. Diese sind (u. a.) in der Form definiert:

```
min(iterable, *, key=None)
max(iterable, *, key=None)
```

Beide Funktionen akzeptieren den Parameter `iterable` aber nicht als Schlüsselwort.

Vielen Funktionen können beliebig viele Argumente positional oder als Schlüsselwort übergeben werden. Im Allgemeinen wird dies durch die Schlüsselwörter `*args` (positionale Argumente) und `**kwargs` (key word arguments, Schlüsselwortargumente) angezeigt. Der Unterschied wird durch das eine bzw. die beiden Sternchen markiert, die Schlüsselwörter selbst sind austauschbar (wie bei der Funktion `print(*objects)`). Das Schlüsselwort `*args` entspricht zugleich dem Symbol \* in der Funktionsdefinition, d. h. rechts davon dürfen nur Schlüsselwortargumente stehen. Weitere Informationen dazu finden Sie [hier](#).

## 19.3 Aufgaben Funktionen

1.

2.

3.

bool()

### 💡 Lösungen

Aufgabe 1: richtig

Aufgabe 2

```
print(1, 2, 3, sep = "_x_")
```

Aufgabe 3: Die Funktion bool() hat ein optionales Argument object mit dem Standardwert False. Das Argument muss positional übergeben werden.

# 20 Flusskontrolle

## 20.0.1 Abzweigungen

```
# Beispiel: Zahl kleiner als ein Schwellwert

a = 7
if a < 10:
    print( 'Die Zahl', a, 'ist kleiner als 10.')
```

```
# Beispiel: Zahl kleiner als ein Schwellwert mit alternativer Ausgabe

a = 13
if a < 10:
    print( 'Die Zahl', a, 'ist kleiner als 10.')
else:
    print( 'Die Zahl', a, 'ist nicht kleiner als 10.')
```

```
# Beispiel: Zahl im Wertebereich zwischen 10 und 20

a = 1
if a < 20 and a > 10:
    print( 'Die Zahl', a, 'liegt zwischen 10 und 20.')
else:
    print( 'Die Zahl', a, 'liegt nicht zwischen 10 und 20.')
```

```
# Beispiel: Zahl im Wertebereich zwischen 10 und 20 mit verschachtelten Abzweigungen

a = 12
if a > 10:
    print( 'Die Zahl', a, 'ist größer als 10.' )

    if a < 20:
        print( 'Die Zahl', a, 'ist kleiner als 20.' )
        print( 'Damit liegt die Zahl zwischen 10 und 20.' )
    else:
        print( 'Die Zahl', a, 'ist größer als 20 und liegt nicht im gesuchten Wertebereich.' )
else:
    print( 'Die Zahl', a, 'ist kleiner als 10 und liegt nicht im gesuchten Wertebereich.' )
```

```
# Beispiel: Zahl im Wertebereich zwischen 10 und 20 mit elif

a = 112
if a < 20 and a > 10:
    print('Die Zahl', a, 'liegt zwischen 10 und 20.')
elif a < 10:
    print('Die Zahl', a, 'ist kleiner als 10 und liegt nicht im gesuchten Wertebereich.' )
elif a > 20 and a <= 100:
    print('Die Zahl', a, 'ist größer als 20, aber nicht größer als 100.')
elif a > 20 and a <= 1000:
    print('Die Zahl', a, 'ist größer als 20, aber nicht größer als 1000.')
else:
    print('Die Zahl', a, 'liegt nicht zwischen 10 und 20 und ist größer als 1000.')
```

## 20.0.2 Schleifen

- 
-

### 20.0.2.1 while-Schleifen

```
# Beispiel: Erhöhen eines Variablenwertes

# Setze Startwert
a = 5

# Definiere Schleife, welche solange ausgeführt
# wird, wie a kleiner als oder gleich 10 ist
while a <= 10:
    # Anweisungsblock der Schleife:

    # 1. Ausgabe des aktuellen Werts von a
    print('aktueller Wert von a', a)

    # 2. Erhöhung von a um Eins
    a += 1

    # Ausgabe des Wertes nach der Schleife
    print('Wert von a nach der Schleife', a)
```

### Warning ??: Endlosschleife

while-Schleifen führen zu einer Endlosschleife, wenn die Abbruchbedingung nicht erreicht werden kann. Beispielsweise fehlt in der folgenden Schleife eine Möglichkeit für die Laufvariable x den Wert 5 zu erreichen.

```
x = 1

while x < 5:
    print(x)
```

In diesem Fall können Sie die Programmausführung durch Drücken von Strg+ C beenden.

### 20.0.2.2 for-Schleifen

```
# range(start = 1, stop = 5) - step wird nicht übergeben, es gilt der Standardwert step = 1
print(range(1, 5), type(range(1, 5)))
```

## lazy evaluation

```
for i in range(1, 5):
    print(i)
```

```
for i in range(1, 15, 3):
    print(i)
```

```
# Ausgabe der geraden Zahlen 1-10 in eine Liste
print("Liste:", list(range(2, 11, 2)))
```

```
# Ausgabe der ungeraden Zahlen 1-10 in ein Tupel
print("Tupel:", tuple(range(1, 11, 2)))
```

```
for i in range(-5, -1):
    print(i)
```

```
for i in range(-1, -5, -1):
    print(i)
```

```
# Absteigende Folge, aber start ist größer als stop
# Die Ausgabe bleibt leer
print(list(range(5, 0)))

# Mit der Funktion reversed geht es
print(list(reversed(range(0, 5)))) 

# Für einer negativen Schrittweite muss start größer sein als stop
print(list(range(0, 5, -1)))
print(list(range(5, 0, -1)))
```

#### 20.0.2.2.1 Listennotation

```
quadratzahlen = [wert ** 2 for wert in range(10, 0, -1)]
print(quadratzahlen)
```

#### 20.0.2.3 Die Schlüsselwörter break und continue

```
x = 0
while x < 10:

    x += 1

    # keine geraden Zahlen ausgeben
    if x % 2 == 0:
        continue

    # Schleife bei x == 7 beenden
    if x == 7:
        break

print(x)
```

### 20.0.3 Ausnahmebehandlung

Methodenbaustein Einlesen strukturierter Datensätze

```
print(1}
```

```
# Beispiel 1: Division durch Null
print(1 / 0)
```

```
# Beispiel 2: undefinierte Variable
print(undefinierte_variable)
```

Ausnahmebehandlung

```
a = 1
b = 2

try:
    differenz = a - b
except:
    print(f"Die Differenz aus {a} und {b} konnte nicht gebildet werden.")
else:
    print(f"Die Differenz aus {a} und {b} ist {differenz}.")
```

```
a = 1
b = 'abc'

try:
    differenz = a - b
except:
    print(f"Die Differenz aus {a} und {b} konnte nicht gebildet werden.")
else:
    print(f"Die Differenz aus {a} und {b} ist {differenz}.")
```

```
a = 1
b = 'abc'
```

```

try:
    differenz = a - b
except Exception as error:
    print(f"Die Differenz aus {a} und {b} konnte nicht gebildet werden.")
    print(error)
else:
    print(f"Die Differenz aus {a} und {b} ist {differenz}.")

```

## 20.1 Aufgaben Flusskontrolle

1.

2.

- 
- 
- 
- 

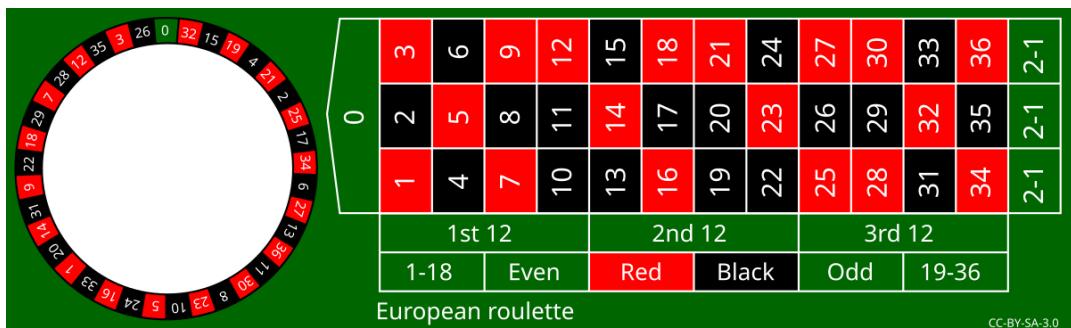


Figure 20.1: Roulette Tableau

CC 3.0 BY-SA

wikimedia.org

## 💡 Musterlösung Aufgaben Flusskontrolle

### 1. Aufgabe

```
# Schreiben Sie ein Programm, das von 1 bis 25 und von 38 bis 50 zählt und jeden Wert, der  
# durch 7 teilbar ist, ausgibt.  
  
i = 0  
while i < 50:  
    i += 1  
    if i > 25 and i < 38 or i % 7 != 0:  
        continue  
    else:  
        print(i)  
  
# Alternative mit Listen  
# Hinweis: wie im Abschnitt "Schleifen" erwähnt muss der Stop-Wert der range-Funktionen um  
numbers = list(range(1,26))+list(range(38,51))  
for i in numbers:  
    if i%7 == 0:  
        print(i)
```

```
7  
14  
21  
42  
49  
7  
14  
21  
42  
49
```

### 2. Aufgabe

```

# Anlegen der zu prüfenden (ganzen) Zahl als Variable
x = 10

# die roten und schwarzen Zahlen werden jeweils als Listen angelegt (ein Tupel funktioniert)
red_numbers = [1,3,5,7,9,12,14,16,18,21,23,25,27,28,30,32,34,36]
black_numbers = [2,4,6,8,10,11,13,15,17,19,20,22,24,26,29,31,33,35]

# die erste if-Abfrage prüft, ob die Zahl überhaupt im Wertebereich liegt. Falls nicht, wird
if x >= 0 and x <= 36:

    # innerhalb der if-Abfrage befinden sich weitere if-Abfragen, die jeweils auf die anderen
    # Prüfungen folgen

    # Prüfung ob die Zahl gerade oder ungerade ist
    if x%2 == 0:
        print("Zahl ist gerade")
    else:
        print("Zahl ist ungerade")

    # Prüfung ob die Zahl rot, schwarz oder grün ist
    if x in red_numbers:
        print("Zahl ist rot")
    elif x in black_numbers:
        print("Zahl ist schwarz")
    elif x == 0:
        print("Zahl ist grün")

    # Prüfung ob die Zahl hoch oder niedrig ist
    if x >= 1 and x <= 18:
        print("Zahl ist niedrig")
    elif x >= 19 and x <= 36:
        print("Zahl ist hoch")

    # Prüfung in welchem Dutzend die Zahl liegt
    if x >= 1 and x <= 12:
        print("Zahl liegt im 1. Dutzend")
    elif x >= 13 and x <= 24:
        print("Zahl ist im 2. Dutzend")
    elif x >= 25 and x <= 36:
        print("Zahl ist im 3. Dutzend")
    else:
        print("Zahl liegt in keinem Dutzend")

else:
    print("Zahl ist nicht im Wertebereich vorhanden.")

```

Zahl ist gerade  
Zahl ist schwarz  
Zahl ist niedrig  
Zahl liegt im 1. Dutzend

Musterlösung von Marc Sönnecken und Maik Poetzsch

# 21 Sammeltypen

- Listen
- Tupel
- Mengen
- Assoziative Arrays

## 21.1 Listen

```
text_variable = 'abc'

liste1 = [1, 'xy', True, text_variable]
print(liste1)

# Listen können auch Listen enthalten
liste2 = [None, liste1]
print(liste2)

# Listen können mit + und * verkettet werden
print(liste1 + liste2)
print(liste1 * 2)
```

```
leere_liste = []
print(leere_liste)
```

## 21.1.1 Slicing: der Zugriffsoperator []

### 21.1.1.1 Zugriff auf einzelne Elemente

```
print(liste1)
print(liste1[0])
print(liste1[3])
```

```
print(liste2)
print(liste2[1])
print(liste2[1][0], liste2[1][1], liste2[1][2], liste2[1][3])
```

```
print(liste1)
print(liste1[-1], liste1[-3])
```

#### 21.1.1.2 Zugriff auf mehrere Elemente

### 21.1.1.3 Zeichenfolgen

```
print('Ich bin ein string'[::-2])
print('Hallo Welt'[0:6])
print('abc'[:-1])
```

## 21.1.2 Listenmethoden

### 21.1.2.1 Elemente bestimmen

- 
- 
- 
- 

```
print(liste1)

liste1.reverse()
print(liste1)

# True wird als 1 gezählt
print("True wird als 1 gezählt:", liste1.index(1), liste1.count(1))
```

### 21.1.2.2 Elemente einfügen

- 
- 
- 

```
print(liste1, "\n")

liste1.append('Hallo')
liste1.extend(['Hallo', 'Welt!'])
liste1.insert(2, '12345')

print(liste1)
```

### 21.1.2.3 Elemente entfernen

- 
- 
- 

```
liste1.remove('Hallo')
print(liste1)

liste1.pop(2)
```

#### 21.1.2.4 Listen und Listenelemente kopieren

```
# Kopieren durch Zuweisung
liste1 = [1, 'xy', True, text_variable]
print("liste1:", liste1, "\n")
liste2 = liste1

## Ändern eines Elements in liste2
liste2[0] = 'ABC'
print("Auch liste1 hat sich durch die Zuweisung in liste2 verändert:", liste1, "\n")
```

```
# Verwendung der Methode liste.copy()
liste1 = [1, 'xy', True, text_variable]
liste2 = liste1.copy()

## Ändern eines Elements in liste2
liste2[0] = 'ABC'
print("liste1 bleibt durch die Zuweisung in liste2 unverändert:", liste1, "\n")

# Verwendung des Slice Operators
liste1 = [1, 'xy', True, text_variable]
liste2 = liste1[:]

## Ändern eines Elements in liste2
liste2[0] = 'ABC'
print("liste1 bleibt durch die Zuweisung in liste2 unverändert:", liste1)
```

```
# Verwendung des Slice Operators
liste1 = [1, 'xy', True, text_variable]
liste2 = liste1[0:2]

# Ändern eines Elements in liste2
liste2[0] = 'ABC'
print("liste1 bleibt durch die Zuweisung in liste2 unverändert:", liste1)
```

```
liste1 = [1, 'xy', True, text_variable]
liste2 = liste1

print("ID liste1:", id(liste1))
print("ID liste2:", id(liste2))
print("ID liste1 gleich ID liste2:", liste1 is liste2)
```

### ⚠ Warning ??: Identität vs. Wertgleichheit

Der Operator `is` prüft die Identität zweier Objekte und unterscheidet sich dadurch vom logischen Operator `==`, der auf Wertgleichheit prüft. Da `liste1` und `liste2` die gleichen Elemente enthalten, liegen sowohl Identität und Wertgleichheit vor. Der Unterschied von Identität und Wertgleichheit kann anhand eines Werts verdeutlicht werden (Im Code-Beispiel wird eine Syntax-Warnung unterdrückt.).

```
# Wertgleichheit
print(1 == 1.0)
print(liste1 == liste2, "\n")

# Identität
print(1 is 1.0)
print(liste1 is liste2)

True
True

False
True
```

### 21.1.3 Aufgaben Listen

1.

2.

3.

•  
•

4.

5.

💡 Tip ??: Musterlösung Listen

1. Aufgabe

```
# 1.  
zahlen = [34, 12, 0, 67, 23]  
gesuchter_index = zahlen.index(0)  
print(gesuchter_index)  
  
zahlen.remove(0)  
zahlen.sort()  
print(zahlen)
```

2  
[12, 23, 34, 67]

2. Aufgabe

```
# 2.  
print(f"Wert der Zahl an Indexposition 1: {zahlen[1]}")  
print(f"Wert der Zahl an Indexposition 3: {zahlen[3]}")
```

Wert der Zahl an Indexposition 1: 23  
Wert der Zahl an Indexposition 3: 67

3. Aufgabe

```
# 3.  
liste = list(range(1, 11))  
print(liste[1::2])  
print(liste[-2::-2])
```

[2, 4, 6, 8, 10]  
[9, 7, 5, 3, 1]

4. Aufgabe

```
# 4.  
# Aufgabe Wochentage  
wochentage = ["Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag", "Samstag", "Sonntag"]  
wochenende = wochentage[5:7] # hier ist es auch möglich wochentage[-2:] zu benutzen  
print("Wochenende:", wochende)  
  
wochentage.pop(-1)  
wochentage.pop(-1)  
print(wochentage)
```

```
Wochenende: ['Samstag', 'Sonntag']  
['Montag', 'Dienstag', 'Mittwoch', 'Donnerstag', 'Freitag']
```

### 5. Aufgabe

```
# 5.  
wochenende.insert(0, wochentage.pop(4))  
print(wochentage)  
print(wochenende)
```

```
['Montag', 'Dienstag', 'Mittwoch', 'Donnerstag']  
['Freitag', 'Samstag', 'Sonntag']
```

Musterlösung von Marc Sönnecken

## 21.2 Tupel

```
tupel1 = (2, 7.8, 'Feuer', True, text_variable)  
tupel2 = (1, )  
  
print(tupel1)
```

```
print(tupel1[2:4])
print(tupel1[::-2])
print(tupel1[-1])
print(tupel1[2:4] + tupel2)
print(3 * tupel2)
```

### 21.2.1 Tupel kopieren

```
# Kopieren durch Zuweisung
tupel1 = (1, 2, 3)
tupel2 = tupel1

## Neuzuweisung der Werte von tupel1
tupel1 = (4, 5, 6)
print(f"Die in tupel2 gespeicherten Werte sind unverändert:\n{tupel1} {tupel2}\n")

# Kopieren mit Slice Operator
tupel1 = (1, 2, 3)
tupel2 = tupel1[:]
print(tupel2 is tupel1)

## Neuzuweisung der Werte von tupel1
tupel1 = (4, 5, 6)
print(tupel1, tupel2)
```

## 21.3 Mengen

```
liste = [1, 1, 5, 3, 3, 4, 2, 'a', 123, 1000, ('tupel', 5)]
print("Das Objekt liste als Menge:\n", set(liste))

menge = {1, 2, 3, 4, 5, 1000, ('tupel', 5), 'a', 123}
print("Die Menge kann auch mit geschweiften Klammern erzeugt werden:", menge, sep = "\n")
```

```
menge_a = set('Python')
menge_b = set('ist super')

# einzigartige Zeichen in a
print("Menge a:", menge_a)

# Zeichen in a, aber nicht in b
print("Menge a - b:", menge_a - menge_b)

# Zeichen in a oder b
print("Menge a | b:", menge_a | menge_b)

# Zeichen in a und b
print("Menge a & b:", menge_a & menge_b)

# Zeichen in a oder b, aber nicht in beiden (XOR)
print("Menge a ^ b:", menge_a ^ menge_b)
```

### 21.3.1 Mengen kopieren

```
# Kopieren durch Zuweisung
set1 = {1, 2, 3}
set2 = set1
print(set1 is set2)

## Neuzuweisen von set1
set1 = {4, 5, 6}
print(f"Die in set2 gespeicherten Werte sind unverändert:\n{set1} {set2}")

# Kopieren durch Methode .copy()
set1 = {1, 2, 3}
set2 = set1.copy()
print(set1 is set2)

## Neuzuweisen von set1
set1 = {4, 5, 6}
print(f"Die in set2 gespeicherten Werte sind unverändert:\n{set1} {set2}")
```

## 21.4 Dictionaries

```
dictionary1 = {1: 'abc', 'b': [1, 2, 3], 'c': ('tupel', 5, 6)}
print(dictionary1, "\n")

print("Werte des Schlüssels 1:", dictionary1[1])
print("Werte des Schlüssels 'b':", dictionary1['b'])
```

```
print("Schlüssel:", dictionary1.keys(), "\n")
print("Werte:", dictionary1.values())
```

#### 21.4.1 Dictionaries kopieren

```
# Kopieren durch Zuweisung
print("dictionary:", dictionary1, "\n")
dictionary2 = dictionary1

## Ändern eines Elements in dictionary2
dictionary2[1] = 'ABC'
print("Auch dictionary1 hat sich durch die Zuweisung in dictionary2 verändert:\n",
      dictionary1, "\n")
```

```
# Verwendung der Methode dictionary.copy()
dictionary1 = {1: 'abc', 'b': [1, 2, 3], 'c': ('tupel', 5, 6)}
dictionary2 = dictionary1.copy()

## Ändern eines Elements in dictionary2
dictionary2[1] = 'ABC'
print("dictionary1 bleibt durch die Zuweisung in dictionary2 unverändert:\n",
      dictionary1, "\n")
```

## 21.5 Übersicht Sammeltypen

## 21.6 Löschen: das Schlüsselwort `del`

```
# Löschen einer Liste
del liste1

# Löschen eines Indexbereichs aus einer Liste
print("Liste vor dem Löschen:", liste2)
del liste2[1:3]
print("Liste nach dem Löschen:", liste2)

# Löschen eines Schlüsselworts aus einem Dictionary
print("Dictionary vor dem Löschen", dictionary1)
del dictionary1[1]
print("Dictionary nach dem Löschen", dictionary1)
```

## 21.7 Funktionen

```
dictionary = {1: 'Kater', 2: 'Fähe', 3: 'Ricke'}
print( liste := list(dictionary) )
print( menge := set(liste) )
print( tupel := tuple(menge) )
```

•  
•

## 21.8 Operationen: Verwendung von Schleifen

**Listing 21.1**

```
zahlen = list(range(1, 11))

quadratzahlen = [] # die Liste muss vor der Schleife angelegt werden
modulo_3 = [] # leere Liste vor der Schleife anlegen

for zahl in zahlen:
    quadratzahl = zahl ** 2
    quadratzahlen.append(quadratzahl)
    modulo_3.append(quadratzahl % 3 == 0)

print(quadratzahlen)
print(modulo_3)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[False, False, True, False, False, True, False, True, False]
```

## 21.9 Aufgaben Sammeltypen

1.

2.

- 
- 
- 

*input()*

3.

💡 Tip ???: Musterlösung Aufgaben Sammeltypen

1. Ganzahlig durch 3 teilbare Quadratzahlen

```
zahlen = list(range(1, 11))

quadratzahlen = [] # die Liste muss vor der Schleife angelegt werden
modulo_3 = [] # leere Liste vor der Schleife anlegen

for zahl in zahlen:
    quadratzahl = zahl ** 2
    if quadratzahl % 3 == 0:
        quadratzahlen.append(quadratzahl)

print(quadratzahlen)
```

[9, 36, 81]

2. Umrechnung von Geschwindigkeiten

```

# Freie Eingabe
## start = int(input("Startwert in Kilometer pro Stunde eingeben."))
## ende = int(input("Endwert in Kilometer pro Stunde eingeben."))
## ausgabeschritte = int(input("Anzahl auszugebener Schritte ein geben."))

# Fixe Werte für die Lösung
start = 5
ende = 107
ausgabeschritte = 8

# Liste für km erstellen
schrittweite = (ende - start) / (ausgabeschritte - 1)
liste_km = []
for i in range(ausgabeschritte):
    liste_km.append(round(start + i * schrittweite))

# Umrechnung
# meter = 1000 * kilometer
# Sekunde = Stunde * 60 * 60
liste_m = []
for wert in liste_km:
    liste_m.append(round((wert * 1000) / (60 * 60), 2))

# Ausgabe
print(f"Schrittweite: {schrittweite:.2f}")
print("Kilometer pro Stunde")
print(liste_km)
print("Meter pro Sekunde")
print(liste_m)

```

Schrittweite: 14.57  
 Kilometer pro Stunde  
 [5, 20, 34, 49, 63, 78, 92, 107]  
 Meter pro Sekunde  
 [1.39, 5.56, 9.44, 13.61, 17.5, 21.67, 25.56, 29.72]

### 3. Sortieren: Bubble Sort Algorithmus

```

# statische Liste, Textausgabe
meine_liste = list(range(9, 0, -1))

if len(meine_liste) > 1:

    print("Liste zu Beginn\t\t : ", meine_liste)

    # äußere Schleife
    Schritt = 0
    for i in range(len(meine_liste) - 1):

        # innere Schleife
        for j in range(len(meine_liste) - 1):
            if meine_liste[j] > meine_liste[j + 1]:
                meine_liste[j], meine_liste[j + 1] = meine_liste[j + 1], meine_liste[j]

        Schritt += 1
        print("Liste nach Schritt ", Schritt, ":", meine_liste)

    print("\nListe sortiert:", *meine_liste) # * unterdrückt die Kommas zwischen den Listen

else:
    print("Die Liste muss mindestens zwei Elemente enthalten!")

```

```

Liste zu Beginn      : [9, 8, 7, 6, 5, 4, 3, 2, 1]
Liste nach Schritt  1 : [8, 7, 6, 5, 4, 3, 2, 1, 9]
Liste nach Schritt  2 : [7, 6, 5, 4, 3, 2, 1, 8, 9]
Liste nach Schritt  3 : [6, 5, 4, 3, 2, 1, 7, 8, 9]
Liste nach Schritt  4 : [5, 4, 3, 2, 1, 6, 7, 8, 9]
Liste nach Schritt  5 : [4, 3, 2, 1, 5, 6, 7, 8, 9]
Liste nach Schritt  6 : [3, 2, 1, 4, 5, 6, 7, 8, 9]
Liste nach Schritt  7 : [2, 1, 3, 4, 5, 6, 7, 8, 9]
Liste nach Schritt  8 : [1, 2, 3, 4, 5, 6, 7, 8, 9]

```

```
Liste sortiert: 1 2 3 4 5 6 7 8 9
```

## 22 Eigene Funktionen definieren

- 
- 
- 

### 22.1 Syntax

```
def Funktionsname(Parameter1, Parameter2):  
    Anweisungsblock  
    return Rückgabewert
```

```
# Beispiel 1: Summe der Quadrate  
  
# Definition einer Funktion zur Berechnung der Summe der Quadrate von zwei Argumenten  
def sum_quadrat(a, b):  
    print('Argument a:', a)
```

```

print('Argument b:', b)
print(18 * '=')
summe = a**2 + b**2
return summe
print("Anweisungen nach dem Schlüsselwort return werden nicht mehr ausgeführt.")

print(sum_quadrat(6, 7))

```

```

ergebnis = sum_quadrat(6, 7)
print(ergebnis)

```

## 22.2 Optionale Parameter

```

# Beispiel 2: optionale Argumente

# Definition einer Funktion zur Berechnung der Summe der Quadrate von zwei Argumenten
def sum_quadrat(a, b, ausgabe = False):
    if ausgabe:
        print('Wert Argument a:', a)
        print('Wert Argument b:', b)
        print(18 * '=')
    summe = a**2 + b**2
    return summe

```

```
print(sum_quadrat(42, 7), "\n")
print(sum_quadrat(42, 7, ausgabe = True))
```

```
# Beispiel 3: mehrere optionale Argumente

# Definition einer Funktion zur Berechnung der Summe der Quadrate von zwei Argumenten
def sum_potenzen(a, b, p = 2, ausgabe = False):
    if ausgabe:
        print('Argument a:', a)
        print('Argument b:', b)
        print('Argument p:', p)
        print(18 * '=')
    summe = a**p + b**p
    return summe

# positionale Übergabe
print(sum_potenzen(42, 7, 3, True), "\n")

# Übergabe per Schlüsselwort
print(sum_potenzen(42, 7, ausgabe = True, p = 4))
```

## 22.3 Rückgabewert(e)

```
# Beispiel 4: mehrere Rückgabewerte

# Definition einer Funktion zur Berechnung der Summe der Quadrate von zwei Argumenten
def sum_potenzen(a, b, p = 2, ausgabe = False):
    if ausgabe:
        print('Argument a:', a)
        print('Argument b:', b)
        print('Argument p:', p)
        print(18 * '=')
    summe = a**p + b**p
    return a, b, summe

ergebnis = sum_potenzen(2, 7, ausgabe = False, p = 4)
print(ergebnis, type(ergebnis))
```

```
print(ergebnis[2])

summe_potenzen = sum_potenzen(2, 7, ausgabe = False, p = 4)[2]
print(summe_potenzen, type(summe_potenzen))
```

## 22.4 Aufgaben Funktionen definieren

1.

2.

3.

•  
•  
•  
•

4.

•  
•  
•  
•  
•

•

## 💡 Musterlösung Aufgaben Funktionen definieren

### 1. Aufgabe Palindrom

```
# 1.  
def is_palindrome(text):  
    text = text.lower()  
  
    return text == text[::-1]  
  
# zu prüfende Zeichenkette in dieser Variablen anlegen  
xyz = "Lagerregal"  
if is_palindrome(xyz) == True:  
    print(f"Ja, {xyz} ist ein Palindrom")  
else:  
    print(f"Nein, {xyz} ist kein Palindrom")
```

Ja, Lagerregal ist ein Palindrom

### 2. Aufgabe Fibonacci

```
# 2.  
def fibonacci(n):  
  
    # die Sonderfälle müssen beachtet werden:  
    if n <= 0:  
        return []  
    elif n == 1:  
        return [0]  
  
    # nachdem die beiden Sonderfälle berücksichtigt wurden, sieht die Liste zum Start immer  
    fibonacci_reihe = [0,1]  
  
    while len(fibonacci_reihe) < n:  
        naechste_zahl = fibonacci_reihe[-1] + fibonacci_reihe[-2]  
        fibonacci_reihe.append(naechste_zahl)  
  
    return fibonacci_reihe  
  
fibonacci(8)
```

[0, 1, 1, 2, 3, 5, 8, 13]

3. Aufgabe Verschlüsselung

## 22.5 mit Listen

```

# 3.

def verschluesseln(str):

    # in sechs Variablen wird einmal das normale und einmal das umgekehrte Alphabet sowie die Ziffern gespeichert
    abc = "abcdefghijklmnopqrstuvwxyz"
    abc_2 = "zyxwvutsrqponmlkjihgfedcba"
    ABC = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    ABC_2 = "ZYXWVUTSRQPONMLKJIHGFEDCBA"
    ziffern = "0123456789"
    ziffern_2 = "9876543210"

    neuer_string = ""

    for i in str:
        if i in ABC: # Großbuchstaben
            position = ABC.index(i)
            neuer_buchstabe = ABC_2[position]

            neuer_string += neuer_buchstabe
        elif i in abc: # Kleinbuchstaben
            position = abc.index(i)
            neuer_buchstabe = abc_2[position]

            neuer_string += neuer_buchstabe
        elif i in ziffern: # Ziffern
            position = ziffern.index(i)
            neuer_buchstabe = ziffern_2[position]

            neuer_string += neuer_buchstabe
        else: # sonstige Zeichen werden unverändert übernommen
            neuer_string += i

    return neuer_string

# zu verschlüsselnde Nachricht:
geheime_nachricht = 'Dies ist das Geheimnis: <*))><'
enkoderter_text = verschluesseln(geheime_nachricht)
print(enkoderter_text)

# die einfachste Möglichkeit die verschlüsselte Nachricht wieder zu entschlüsseln ist, die
dekoderter_text = verschluesseln(enkoderter_text)
print(dekoderter_text)

```

```
Wrvh rhg wzh Tvsrvnmrh: <*>))><
Dies ist das Geheimnis: <*>))><
```

## 22.6 mit `ord()` und `chr()`

```
def verschluesseln(a):
    a = list(a)
    # print(a)
    b = [] # output list

    for i in a:
        # print(i, ord(i))
        b.append(ord(i))
    # print(b)

    # invertieren a = 97, z = 122, A = 65, Z = 90, 0 = 48, 9 = 57
    # Max - Wert + Min

    for i in range(len(b)):
        if b[i] >= 65 and b[i] <= 90: # Großbuchstaben
            b[i] = chr(90 - b[i] + 65)
        elif b[i] >= 97 and b[i] <= 122: # Kleinbuchstaben
            b[i] = chr(122 - b[i] + 97)
        elif b[i] >= 48 and b[i] <= 57: # Zahlen
            b[i] = chr(57 - b[i] + 48)
        else: # sonstige Zeichen
            b[i] = chr(b[i])

    return ''.join(b)

print(enkodierter_text)
print(verschluesseln(a = enkodierter_text))
```

```
Wrvh rhg wzh Tvsrvnmrh: <*>))><
Dies ist das Geheimnis: <*>))><
```

4. Temperaturkonverter

```

# 4.

def temperatur_umrechnen(wert, von_einheit, nach_einheit):

    # if-Abfragen um zu prüfen, welche Einheit der umzuwandelnde Wert hat. Je nachdem welche

    # if-Abfrage für Ursprungswert in Celsius:
    if von_einheit == "C":
        if nach_einheit == "F":
            return wert * 9/5 + 32
        elif nach_einheit == "K":
            return wert + 273.15
        elif nach_einheit == "C":
            return wert

    # if-Abfrage für Ursprungswert in Fahrenheit:
    if von_einheit == "F":
        if nach_einheit == "C":
            return (wert - 32) * 5/9
        elif nach_einheit == "K":
            return (wert - 32) * 5/9 + 273.15
        elif nach_einheit == "F":
            return wert

    # if-Abfrage für Ursprungswert in Kelvin:
    if von_einheit == "K":
        if nach_einheit == "C":
            return wert - 273.15
        elif nach_einheit == "F":
            return (wert - 273.15) * 9/5 + 32
        elif nach_einheit == "K":
            return wert

    # für ungültige Einheiten wird None zurückgegeben
    return None

test_temp = temperatur_umrechnen(37, "C", "F")
print(f"37°C sind in Fahrenheit: {test_temp}°F")

```

37°C sind in Fahrenheit: 98.6°F

Musterlösung von Marc Sönnecken und Maik Poetzsch



# 23 Dateien lesen und schreiben

## 23.1 Dateiobjekte

open  
Dateiobjekt

### 23.1.1 Dateipfad

```
pfad_maya = "01-daten/dice-maya.txt"  
pfad_hans = "01-daten/dice-hans.txt"
```

💡 Tip ???: Arbeitsverzeichnis in Python ermitteln und wechseln

Der Pfad des aktuellen Arbeitsverzeichnisses kann mit dem Modul `os` mittels `os.getcwd()` ermittelt werden (hier ohne Ausgabe). Mit `os.chdir('neuer_pfad')` kann das Arbeitsverzeichnis ggf. gewechselt werden. Die korrekte Formatierung des Pfads erkennen Sie an der Ausgabe von `os.getcwd()`.

```
import os  
print(os.getcwd())
```

Das Importieren von Modulen wird in einem späteren Kapitel behandelt.

### 23.1.2 Zugriffsmodus

[Dokumentation](#)

```
daten_maya = open(pfad_maya, mode = 'r')  
print(daten_maya)
```

```
daten_maya = open(pfad_maya, mode = 'r', encoding = 'UTF-8')  
print(daten_maya)
```

### Note ??: Attribute eines Objekts bestimmen

Mit der Funktion `dir(objekt)` können die verfügbaren Attribute eines Objekts ausgegeben werden. Dabei werden jedoch auch die vererbten Attribute und Methoden der Klasse des Objekts ausgegeben, sodass die Ausgabe oft sehr umfangreich ist. Zum Beispiel für die Ganzzahl 1:

```
print(dir(1))
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__di...
```

Um die Ausgabe auf Attribute einzuschränken, kann folgende Funktion verwendet werden:

```
objekt = 1
```

```
attribute = [attr for attr in dir(objekt) if not callable(getattr(objekt, attr))]
print(attribute)
```

```
['__doc__', 'denominator', 'imag', 'numerator', 'real']
```

Mit doppelten Unterstrichen umschlossene Attribute sind für Python reserviert und nicht für den:die Nutzer:in gedacht. Folgende Funktion entfernt Attribute mit doppelten Unterstrichen aus der Ausgabe:

```
objekt = 1
```

```
attribute = [attr for attr in dir(objekt) if not (callable(getattr(objekt, attr)) or attr.s...
```

```
['denominator', 'imag', 'numerator', 'real']
```

Im Fall einer Ganzzahl können Attribute (zur Abgrenzung von Gleitkommazahlen in umschließenden Klammern) wie folgt aufgerufen werden:

```
(1).numerator
```

```
1
```

Wenn wir uns die Attribute des Dateiobjekts ‘daten\_maya’ ansehen, fallen Attribute mit einem einzelnen führenden Unterstrich auf.

```
objekt = daten_maya

attribute = [attr for attr in dir(objekt) if not (callable(getattr(Objekt, attr)) or attr.s
print(attribute)
```

```
['_CHUNK_SIZE', '_finalizing', 'buffer', 'closed', 'encoding', 'errors', 'line_buffering',
```

Hierbei handelt es sich um Attribute, die nicht durch den Nutzer:in aufgerufen werden sollen (weitere Informationen dazu finden Sie [hier](#)). Folgender Programmcode gibt alle Attribute ohne führende Unterstriche aus:

```
objekt = daten_maya

attribute = [attr for attr in dir(objekt) if not (callable(getattr(Objekt, attr)) or attr.s
print(attribute)
```

```
['buffer', 'closed', 'encoding', 'errors', 'line_buffering', 'mode', 'name', 'newlines', 'w
```

```
print(f"Dateipfad: {daten_maya.name}\n"
      f"Dateiname: {os.path.basename(daten_maya.name)}\n"
      f"Datei ist geschlossen: {daten_maya.closed}\n"
      f"Zugriffsmodus: {daten_maya.mode}\n"
      f"Enkodierung: {daten_maya.encoding}")
```

### 💡 Tip ??: Rückfalloption

In der Datenanalyse werden in der Regel spezialisierte Pakete wie NumPy oder Pandas verwendet. Diese vereinfachen das Einlesen von Dateien gegenüber der Pythonbasis erheblich. Dennoch ist es sinnvoll, sich mit den Methoden der Pythonbasis zum Einlesen von Dateien vertraut zu machen. Denn das Einlesen mit der Funktion `open()` klappt so gut wie immer - es ist eine gute Rückfalloption.

### 23.1.3 Dateiinhalt ausgeben

```
i = 0
for zeile in daten_maya:
    print(f"Inhalt Zeile {i}, mit {len(zeile)} Zeichen:")
    print(zeile)
    i += 1
```

## 23.2 Dateien einlesen

datenobjekt.read()

```
augen_maya = daten_maya.read()

print(f"len(augen_maya): {len(augen_maya)}\n\n"
      f"Inhalt der Datei augen_maya:\n{augen_maya}")
```

### ⚠ Warning ???: Dateizeiger in Python

Wird eine Datei zeilenweise oder mit der Methode `.read()` ausgelesen, wird der Dateizeiger um die angegebene Zeichenzahl bzw. bis ans Ende der Datei bewegt. Wird beispielsweise ein Datensatz ‘daten’ geöffnet und mit der Methode `daten.read(3)` die ersten drei Zeichen ausgelesen, bewegt sich der Dateizeiger von der Indexposition 0 zur Indexposition 3 (bzw. steht jeweils davor).

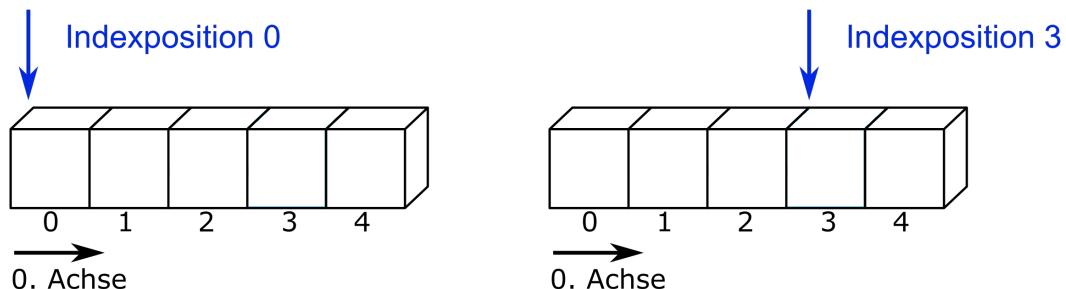


Figure 23.1: Bewegung des Dateizeigers beim Auslesen von drei Zeichen

Die Methode `daten.tell()` gibt zurück, an welcher Position sich der Dateizeiger befindet.

Mit der Methode `daten.seek(offset, whence = 0)` wird der Zeiger an eine bestimmte Position gesetzt. Die Methode akzeptiert das Argument `offset` (Versatz) und das optionale Argument `whence` (woher), dessen Standardwert 0 (Dateianfang) ist. Für Zugriffe im **Binärmodus** (`open(pfad, mode = 'rb')`) kann das Argument `whence` außerdem die Werte 1 (aktuelle Position) oder 2 (Dateiende) annehmen.

- `daten.seek(0, 0)` bezeichnet den Dateianfang
- `daten.seek(0, 1)` bezeichnet die aktuelle Position in der Datei
- `daten.seek(0, 2)` bezeichnet das Dateiende
- `daten.seek(-3, 2)` bezeichnet das dritte Zeichen vor dem Dateiende

```
print(f"Position des Dateizeigers vor dem Zurücksetzen auf 0: {daten_maya.tell()}")  
  
daten_maya.seek(0);  
print(f"Position des Dateizeigers nach dem Zurücksetzen auf 0: {daten_maya.tell()}")
```

```
augen_maya = daten_maya.read()

print(f"Inhalt des Objekts augen_maya:\n{augen_maya}")
```

💡 Tip ???: Musterlösung Dateizeiger bewegen

```
daten_maya.seek(6, 0);
print(daten_maya.read(1))

daten_maya.seek(daten_maya.tell() + 4, 0);
print(daten_maya.read(1))
```

```
6
2
```

```
print(type(augen_maya))
```

```
print(f"augen_maya:\n{augen_maya}")

augen_maya = augen_maya.strip(' ')
print(f"\naugen_maya.strip(' '):\n{augen_maya}")
```

```
augen_maya = augen_maya.split('', '')
print(f"\naugen_maya.split('\'', \'\'):\\n{augen_maya}")

augen_maya_int = []
for i in augen_maya:
    augen_maya_int.append(int(i))

print(f"\naugen_maya_int:\\n{augen_maya_int}\\n\\nSumme Augen: {sum(augen_maya_int)}")
```

### 23.2.0.1 Datei schließen

```
daten_maya.close()
```

#### ⚠ Warning ??: Schreiboperationen mit Python

Das Schließen einer Datei ist besonders für Schreiboperationen auf Datenobjekten wichtig. Andernfalls kann es passieren, dass Inhalte mit `datenobjekt.write()` nicht vollständig auf den Datenträger geschrieben werden. Siehe dazu die [Dokumentation](#).

## 23.3 Aufgabe Dateien einlesen

💡 Tip ??: Musterlösung Augenzahlvergleich

```
# Erst Einlesen der Datei:  
daten_hans = open(pfad_hans, mode = 'r', encoding = 'UTF-8')  
augen_hans = daten_hans.read()  
print(augen_hans)  
# Hier muss man erkennen, dass Hans seinen Namen an den Anfang seiner Liste gesetzt hat. Da  
# er dann eine leere Zeichenkette vor den Zahlen hat.  
  
augen_hans = augen_hans.strip('"Hans", ')  
augen_hans = augen_hans.strip('')  
augen_hans = augen_hans.split(", ")  
print(augen_hans)  
# print-Ausgabe zeigt, dass die Liste nun korrekt bereinigt wurde. Sie besteht nur noch aus  
# den Würfelergebnissen.  
  
# Neue (leere) Liste für die Würfe von Hans anlegen:  
augen_hans_int = []  
for i in augen_hans:  
    augen_hans_int.append(int(i))  
  
print(f"Summe Augenzahl von Hans: {sum(augen_hans_int)}")  
  
"Hans", "3", "5", "1", "3", "2", "5"  
['3', '5', '1', '3', '2', '5']  
Summe Augenzahl von Hans: 19
```

Musterlösung von Marc Sönnecken.

## 23.4 Daten interpretieren

String-Methoden

## 23.5 for-Schleife mit break

```
dateipfad = "01-daten/einwohner_europa_2019.csv"
dateiobjekt_einwohner = open(dateipfad, 'r')

# erste 3 Zeilen anschauen
i = 0
for zeile in dateiobjekt_einwohner:

    print(zeile)
    i += 1
    if i == 3:
        break

# Datei schließen
dateiobjekt_einwohner.close()
```

## 23.6 Methode dateiobjekt.readline()

```
dateipfad = "01-daten/einwohner_europa_2019.csv"
dateiobjekt_einwohner = open(dateipfad, 'r')

for i in range(3):
    print(dateiobjekt_einwohner.readline())

# Datei schließen
dateiobjekt_einwohner.close()
```

```
dateipfad = "01-daten/einwohner_europa_2019.csv"
dateiobjekt_einwohner = open(dateipfad, 'r')

einwohner = dateiobjekt_einwohner.read()
print(einwohner)

# Datei schließen
dateiobjekt_einwohner.close();
```

```
liste_einwohner_zeilenweise = einwohner.split("\n")
print(liste_einwohner_zeilenweise[0:3])
```

```
spaltennamen = liste_einwohner_zeilenweise.pop(0)
spaltennamen = spaltennamen.split(',')
print(f"Überschrift Spalte 0: {spaltennamen[0]}\tÜberschrift Spalte 1: {spaltennamen[1]}")
```

```

# Leere Listen vor der Schleife anlegen
geo = []
einwohnerzahl = []

try:
    for zeile in liste_einwohner_zeilenweise:
        eintrag = zeile.split(',')
        geo.append(eintrag[0])
        einwohnerzahl.append(eintrag[1])

    print(spaltennamen[0])
    print(geo, "\n")

    print(spaltennamen[1])
    print(einwohnerzahl)

except Exception as error:
    # print Fehlermeldung
    print(f"Fehlermeldung: {error}")

    # print Eintrag und Index
    print(f"Eintrag: {eintrag}\t Zeilenindex: {liste_einwohner_zeilenweise.index(zeile)}")

```

```

# leere Zeile entfernen
liste_einwohner_zeilenweise.remove('')

# Leere Listen vor der Schleife anlegen
geo = []
einwohnerzahl = []

try:
    for zeile in liste_einwohner_zeilenweise:
        eintrag = zeile.split(',')
        geo.append(eintrag[0])
        einwohnerzahl.append(eintrag[1])

        print(spaltennamen[0])
        print(geo, "\n")

        print(spaltennamen[1])
        print(einwohnerzahl)

except IndexError as error:
    print(error)

```

## 23.7 Aufgabe Daten interpretieren

- 1.
- 2.
- 3.
- 
-

💡 Musterlösung vollständiges Einlesen

```
# 1. Minimum und Maximum der Einwohnerzahlen und dazugehörige Länder bestimmen
liste_einwohner = []
for ele in einwohnerzahl:
    liste_einwohner.append(int(ele))

# Umwandeln der Strings in Integer bei der Einwohnerzahl
einwohnerzahl = [int(zahl) for zahl in einwohnerzahl]

# 2. Fehler bereinigen
# Sowohl der Fehler in der Liste der Einwohnerzahlen als auch das entsprechende Land werden
for i in range(len(einwohnerzahl) -1,-1,-1): # Hiermit wird rückwärts durch die Liste iteriert
    if einwohnerzahl[i] < 0:
        einwohnerzahl.pop(i)
        geo.pop(i)

for i in range(len(einwohnerzahl)):
    if einwohnerzahl[i] == max(einwohnerzahl):
        maximum = i
    elif einwohnerzahl[i] == min(einwohnerzahl):
        minimum = i

maximum_land = geo[maximum]
minimum_land = geo[minimum]
print(f"Maximum: {maximum_land} mit {max(einwohnerzahl)} Einwohnern")
print(f"Minimum: {minimum_land} mit {min(einwohnerzahl)} Einwohnern")
# 3. Einwohner Europa insgesamt

pop_gesamt = sum(einwohnerzahl) # Uganda gehört nicht zu Europa, wurde durch die Fehlerbehandlung entfernt
print(f"Die Summe aller Einwohner in Europa beträgt {pop_gesamt}")

# 4.
print(f"Die Liste einwohnerzahl hat den Datentypen {type(einwohnerzahl)}")
print(f"Die Einträge der Liste haben den Datentypen {type(einwohnerzahl[0])}")
```

Maximum: Deutschland einschliesslich ehemalige DDR mit 82940663 Einwohnern  
Minimum: Malta mit 493559 Einwohnern

Die Summe aller Einwohner in Europa beträgt 514117784

Die Liste einwohnerzahl hat den Datentypen <class 'list'>

Die Einträge der Liste haben den Datentypen <class 'int'>

## 23.8 Einlesen als Liste

```
dateipfad = "01-daten/einwohner_europa_2019.csv"
dateiobjekt_einwohner = open(dateipfad, 'r')

# Methode readlines
einwohner = dateiobjekt_einwohner.readlines()
print(einwohner)

## Dateizeiger zurücksetzen
dateiobjekt_einwohner.seek(0);

# Funktion list
einwohner = list(dateiobjekt_einwohner)
print(einwohner)

# Datei schließen
dateiobjekt_einwohner.close();
```

```
print('Hund'.replace('Hu', 'Mu'))  
  
zeichenfolge = 'Ein  kurzer Text ohne  doppelte Leerzeichen.'  
  
print(zeichenfolge.replace('  ', ' '))
```

```
print(zeichenfolge.replace('  ', ' ').replace('doppelte', ''))
```

```
dateipfad = "01-daten/einwohner_europa_2019.csv"  
dateiobjekt_einwohner = open(dateipfad, 'r')  
  
# Methode readlines  
einwohner = dateiobjekt_einwohner.readlines()  
einwohner_neu = []  
  
for element in einwohner:  
    einwohner_neu.append(element.replace('\n', ''))  
  
einwohner = einwohner_neu  
print(einwohner)  
  
# Datei schließen  
dateiobjekt_einwohner.close();
```

## 23.9 Dateien schreiben

```
dateipfad = "01-daten/neue_datei.txt"

# Öffne Datei zum Schreiben öffnen
datei = open(dateipfad, mode = 'w')

# Inhalt in die Datei schreiben
datei.write("Prokrastination an Hochschulen\n\n".upper())
datei.write("KAPITEL 1: Aller Anfang ist schwer\nPlatzhalter: Den Rest schreibe ich später.")

# Datei schließen

datei.close()
```

```
dateiinhalt = open(dateipfad, mode = 'r')
text = dateiinhalt.read()
print(text)

dateiinhalt.close()
```

## 23.10 Aufgabe Dateien schreiben

1.

# 24 Module und Pakete importieren

! Important ??: Module und Pakete

**Module** Module sind Dateien, die Funktionsdefinitionen enthalten.

Module werden durch das Schlüsselwort `import` und ihren Namen importiert, bspw.  
`import glob`.

**Pakete** Pakete sind Sammlungen von Modulen.

In Paketen enthaltene Module werden durch das Schlüsselwort `import` mit der Schreibweise `paket.modul` importiert, bspw. `import matplotlib.pyplot`.

```
import random

print(random.randint(1, 10)) # Zufällige Ganzzahl zwischen 1 und 10
```

```
import matplotlib.pyplot

zufallsdaten = [] # leere Liste anlegen
for i in range(10):
    zufallszahl = random.randint(1, 10)
    zufallsdaten.append(zufallszahl)

matplotlib.pyplot.plot(zufallsdaten)
```

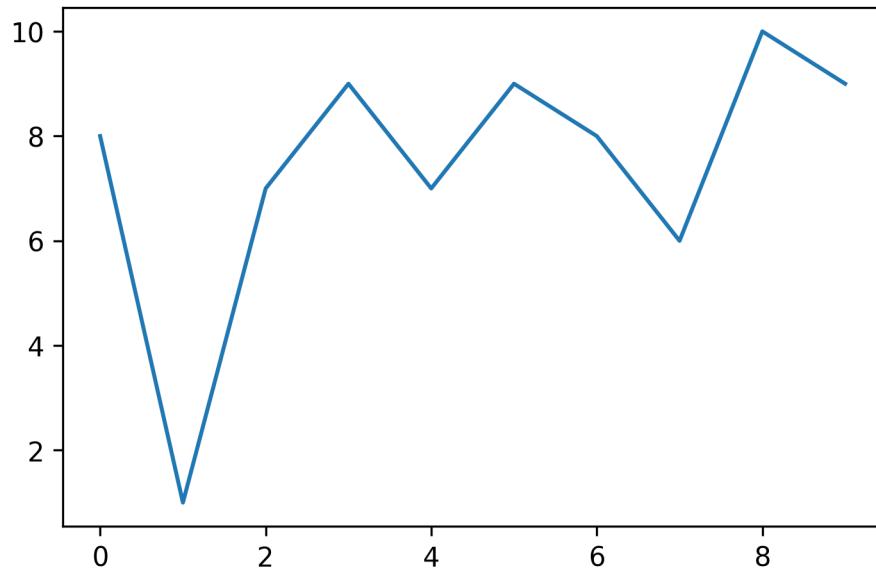


Figure 24.1: Grafik mit dem Modul pyplot aus dem Paket matplotlib

### ⚠ Warning ???: Namensraum direkt einbinden

In Python ist es auch möglich, Funktionen direkt in den Namensraum von Python zu importieren, sodass diese ohne die Schreibweise `modul.funktion()` aufgerufen werden können. Dies ist mit dem Schlüsselwort `from` möglich.

```
from random import randint
print(f"Die Funktion randint steht nun direkt zur Verfügung: {randint(1, 100)}")
```

Die Funktion `randint` steht nun direkt zur Verfügung: 21

Durch `from modulname import *` ist es sogar möglich, alle Funktionen aus einem Modul in den Namensraum von Python zu importieren. Im Allgemeinen sollte das direkte Importieren von Funktionen oder eines ganzen Moduls in den Namensraum von Python jedoch unterlassen werden. Einerseits wird damit eine Namensraumkollision riskiert, beispielsweise gibt es die Funktion `sum()` in der Pythonbasis, in NumPy und in Pandas. Andererseits wird der Programmcode dadurch weniger nachvollziehbar, da nicht mehr überall ersichtlich ist, aus welchem Modul eine verwendete Funktion stammt.

## 24.1 import as

```
import matplotlib.pyplot as plt  
plt.plot(zufallsdaten)
```

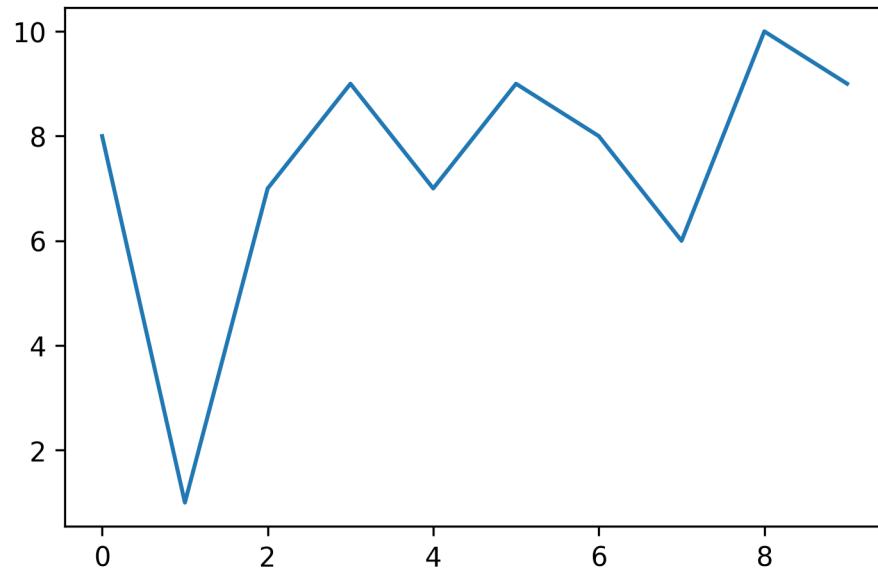


Figure 24.2: Grafik mit dem Modul pyplot aus dem Paket matplotlib

## 24.2 Kleine Modulübersicht

•  
•  
•  
•  
•  
•  
•  
•  
•

## **25 Das Wichtigste**

# **Part IV**

## **w-python-numpy**

# Werkzeugbaustein NumPy



Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch und Sebastian Seipel. "Werkzeugbaustein NumPy" von Marc Fehr und Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar unter <https://github.com/bausteine-der-datenanalyse/w-python-numpy-grundlagen>. Ausgenommen von der Lizenz sind alle Logos und anders gekennzeichneten Inhalte. 2025

<https://github.com/bausteine-der-datenanalyse/w-python-numpy-grundlagen>

# **Intro**

## **Voraussetzungen**

- 
- 
- 

## **Verwendete Pakete und Datensätze**

### **Pakete**

- NumPy
- Matplotlib

### **Datensätze**

- TC01.csv
- Bild: Mona Lisa
- Bild: Campus

## **Bearbeitungszeit**

## **Lernziele**

- 
- 
- 
-

•

# 26 Einführung NumPy

## 26.1 Vorteile & Nachteile

**i** Warum ist numpy oftmals schneller?

NumPy implementiert eine effizientere Speicherung von Listen im Speicher. Nativ speichert Python Listeninhalte aufgeteilt, wo gerade Platz ist.

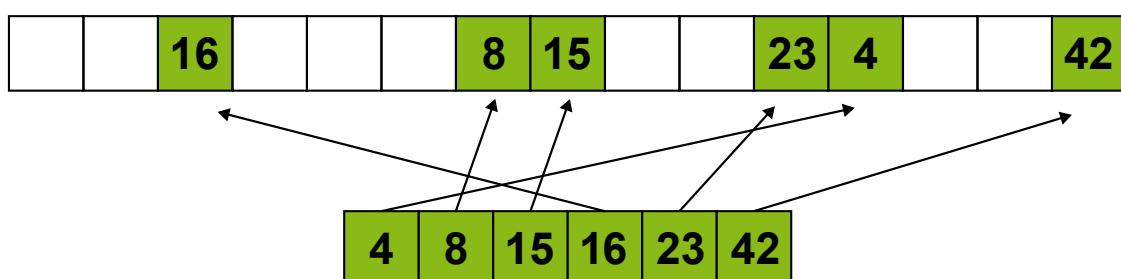


Figure 26.1: Speicherung von Daten in nativem Python

Dagegen werden NumPy-Arrays und Matrizen zusammenhängend gespeichert, was einen effizienteren Datenaufruf ermöglicht.

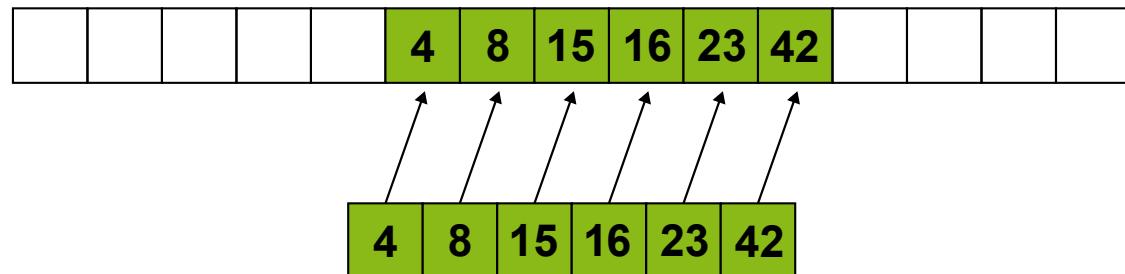


Figure 26.2: Speicherung von Daten bei Numpy

Dies bedeutet aber auch, dass eine Erweiterung einer Liste deutlich schneller ist als eine Erweiterung von Arrays oder Matrizen. Bei Listen kann jeder freie Platz genutzt werden, während Arrays und Matrizen an einen neuen Ort im Speicher kopiert werden müssen.

## 26.2 Einbinden des Pakets

```
import numpy as np
```

## 26.3 Referenzen

[Dokumentation](#)

## 27 Erstellen von NumPy arrays

(1, 2, 3, 4, 5, 6)

und die Matrix

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
liste = [1, 2, 3, 4, 5, 6]

matrix = [[1, 2, 3], [4, 5, 6]]

print(liste)
print(matrix)
```

```
liste = np.array([1, 2, 3, 4, 5, 6])

matrix = np.array([[1, 2, 3], [4, 5, 6]])

print(liste)
print(matrix)
```

```
matrix_3d = np.array([[ [1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
```

```
np.zeros([2,3])
```

```
np.ones([2,3])
```

```
np.full([2,3],7)
```

💡 Wie könnte man Arrays, die mit einer Zahl x gefüllt sind, auch erstellen?

Der Trick besteht hierbei darin, ein Array mit `np.ones()` zu initialisieren und dieses Array dann mit der Zahl x zu multiplizieren. Im folgenden Beispiel ist `x = 5`.

```
np.ones([2,3]) * 5
```

```
array([[5., 5., 5.],  
       [5., 5., 5.]])
```

```
np.linspace(0,1,11)
```

```
np.arange(0,10,2)
```

### 💡 Zwischenübung: Array Erstellung

Erstellen Sie jeweils ein NumPy-Array, mit dem folgenden Inhalt:

1. mit den Werten 1, 7, 42, 99
2. zehn mal die Zahl 5
3. mit den Zahlen von 35 **bis einschließlich** 50
4. mit allen geraden Zahlen von 20 **bis einschließlich** 40
5. eine Matrix mit 5 Spalten und 4 Reihen mit dem Wert 4 an jeder Stelle
6. mit 10 Werten die gleichmäßig zwischen 22 und einschließlich 40 verteilt sind

### Lösung

```
# 1.  
print(np.array([1, 7, 42, 99]))
```

```
[ 1  7 42 99]
```

```
# 2.  
print(np.full(10,5))
```

```
[5 5 5 5 5 5 5 5 5 5]
```

```
# 3.  
print(np.arange(35, 51))
```

```
[35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50]
```

```
# 4.  
print(np.arange(20, 41, 2))
```

```
[20 22 24 26 28 30 32 34 36 38 40]
```

```
# 5.  
print(np.full([4,5],4))
```

```
[[4 4 4 4 4]  
 [4 4 4 4 4]  
 [4 4 4 4 4]  
 [4 4 4 4 4]]
```

```
# 6.  
print(np.linspace(22, 40, 10))
```

```
[22. 24. 26. 28. 30. 32. 34. 36. 38. 40.]
```

## 28 Größe, Struktur und Typ

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
```

```
np.shape(matrix)
```

```
len(matrix)
```

```
np.ndim(matrix)
```

💡 Die Ausgabe von `np.ndim()` kann mit `np.shape()` und einer nativen Python-Funktion erreicht werden. Wie?

`np.ndim()` gibt die Länge der Liste von `np.shape()` aus.

```
len(np.shape(matrix))
```

2

```
np.size(matrix)
```

```
typ_a = np.array([1, 2, 3, 4, 5])
typ_b = np.array([0.1, 0.2, 0.3, 0.4, 0.5])
typ_c = np.array(["Montag", "Dienstag", "Mittwoch"])
```

```
print(typ_a.dtype)
```

```
print(typ_b.dtype)
```

```
print(typ_c.dtype)
```

Table 28.1: Typische Datentypen in NumPy

💡 Zwischenübung: Array-Informationen auslesen

Gegeben sei folgende Matrix:

```
matrix = np.array([[ [ 0,  1,  2,  3],  
                     [ 4,  5,  6,  7],  
                     [ 8,  9, 10, 11]],  
  
                   [[12, 13, 14, 15],  
                    [16, 17, 18, 19],  
                    [20, 21, 22, 23]],  
  
                   [[24, 25, 26, 27],  
                    [28, 29, 30, 31],  
                    [32, 33, 34, 35]])
```

Bestimmen Sie durch Anschauen die Anzahl an Dimensionen und die Länge jeder Dimension. Von welchem Datentyp ist der Inhalt dieser Matrix?

Überprüfen Sie daraufhin Ihre Ergebnisse, indem Sie die passenden NumPy-Funktionen anwenden.

**Lösung**

```
matrix = np.array([[[ 0,  1,  2,  3],
                   [ 4,  5,  6,  7],
                   [ 8,  9, 10, 11]],

                  [[12, 13, 14, 15],
                   [16, 17, 18, 19],
                   [20, 21, 22, 23]],

                  [[24, 25, 26, 27],
                   [28, 29, 30, 31],
                   [32, 33, 34, 35]]])

anzahl_dimensionen = np.ndim(matrix)

print("Anzahl unterschiedlicher Dimensionen: ", anzahl_dimensionen)

laenge_dimensionen = np.shape(matrix)

print("Länge der einzelnen Dimensionen: ", laenge_dimensionen)

print(matrix.dtype)
```

```
Anzahl unterschiedlicher Dimensionen: 3
Länge der einzelnen Dimensionen: (3, 3, 4)
int64
```

# 29 Rechnen mit Arrays

## 29.1 Arithmetische Funktionen

```
a = np.array([1, 2, 3, 4, 5])  
b = np.array([9, 8, 7, 6, 5])
```

```
np.add(a,b)
```

```
a + b
```

```
ergebnis = np.ones(5)  
for i in range(len(a)):  
    ergebnis[i] = a[i] + b[i]  
  
print(ergebnis)
```

- 
- 
- 
- 
- 
- 
- 
- 

### ⚠️ Arbeiten mit Winkelfunktionen

Wie auch am Taschenrechner birgt das Arbeiten mit den Winkelfunktionen ( $\sin$ ,  $\cos$ , ...) die Fehlerquelle, dass man nicht mit Radian-Werten, sondern mit Grad-Werten arbeitet. Die Winkelfunktionen in NumPy erwarten jedoch Radian-Werte.  
Für eine einfache Umrechnung bietet NumPy die Funktionen `np.deg2rad()` und `np.rad2deg()`.

## 29.2 Vergleiche

```
a = np.array([1, 2, 3, 4, 5])  
  
b = np.array([9, 2, 7, 4, 5])
```

```
a == b
```

```
a < b
```

```
a = np.array(0.1 + 0.2)
b = np.array(0.3)
a == b
```

```
a = np.array(0.1 + 0.2)
b = np.array(0.3)
print(np.isclose(a, b, atol=0.001))
```

**i** Warum ist  $0.1 + 0.2$  nicht gleich  $0.3$ ?

Zahlen werden intern als Binärzahlen dargestellt. So wie  $1/3$  nicht mit einer endlichen Anzahl an Ziffern korrekt dargestellt werden kann, müssen Zahlen ggf. gerundet werden, um im Binärsystem dargestellt zu werden.

```
a = 0.1
b = 0.2
print(a + b)
```

```
0.30000000000000004
```

## 29.3 Aggregatfunktionen

```
a = np.array([1, 2, 3, 4, 8])
```

```
np.sum(a)
```

```
np.min(a)
```

```
np.mean(a)
```

```
np.median(a)
```

```
np.std(a)
```

### Zwischenübung: Rechnen mit Arrays

Gegeben sind zwei eindimensionale Arrays a und b:

a = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100]) und b = np.array([5, 15, 25, 35, 45, 55, 65, 75, 85, 95])

1. Erstellen Sie ein neues Array, das die Sinuswerte der addierten Arrays a und b enthält.
2. Berechnen Sie die Summe, den Mittelwert und die Standardabweichung der Elemente in a.
3. Finden Sie den jeweils größten und den kleinsten Wert in a und b.

### Lösung

```
a = np.array([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
b = np.array([5, 15, 25, 35, 45, 55, 65, 75, 85, 95])

# 1.
sin_ab = np.sin(a + b)

# 2.
sum_a = np.sum(a)
mean_a = np.mean(a)
std_a = np.std(a)

# 3.
max_a = np.max(a)
min_a = np.min(a)
max_b = np.max(b)
min_b = np.min(b)
```

# 30 Slicing

## 30.1 Normales Slicing mit Zahlenwerten

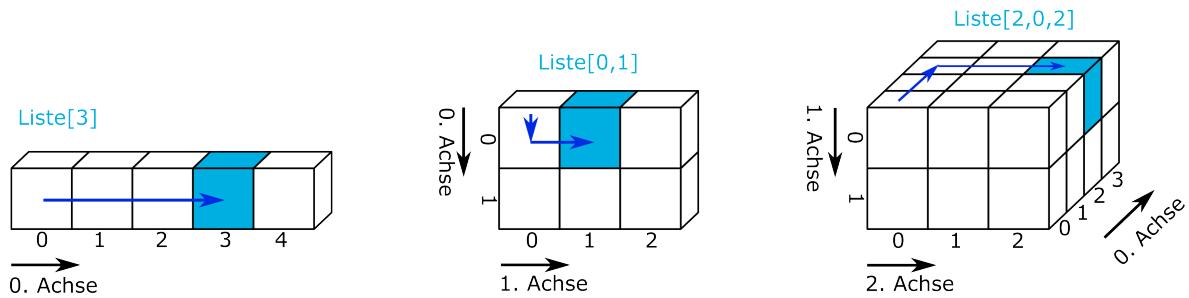


Figure 30.1: Ansprechen der einzelnen Achsen für den ein-, zwei- und dreidimensionalen Fall inkl. jeweiligem Beispiel

- 1.
- 2.
- 3.

```
liste = np.array([1, 2, 3, 4, 5, 6])
```

```
# Auswählen des ersten Elements  
liste[0]
```

```
# Auswählen des letzten Elements  
liste[-1]
```

```
# Auswählen einer Reihe von Elementen  
liste[1:4]
```

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
```

```
# Auswählen eines Elements  
matrix[1,1]
```

```
matrix_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])  
print(matrix_3d)
```

```
# Auswählen eines Elements  
matrix_3d[1,0,2]
```

## 30.2 Slicing mit logischen Werten (boolesche Masken)

```
# Erstellen wir ein Beispiel Array
a = np.array([1, 2, 3, 4, 5, 6])

# Erstellen der Maske
maske = a > 3

print(maske)
```

```
# Anwenden der Maske
print(a[maske])
```

```
# 2 Masken
gerade_zahlen = a % 2 == 0
kleiner_3 = a < 3
print(gerade_zahlen, kleiner_3, sep = '\n')

# logisches UND beider Masken
print(a[kleiner_3 * gerade_zahlen])
```

### ⚠ Warning

Das Verwenden von booleschen Arrays ist nicht für native Python-Listen möglich. Hier muss durch die Liste iterriert werden.

```
a = [1, 2, 3, 4, 5, 6]
ergebnis = [x for x in a if x > 3]
print(ergebnis)
```

[4, 5, 6]

### 💡 Zwischenübung: Array-Slicing

Wählen Sie die farblich markierten Bereiche aus dem Array “matrix” mit den eben gelernten Möglichkeiten des Array-Slicing aus.

0	2	11	18	47	33	48	9	31	8	41
1	55	1	8	3	91	56	17	54	23	12
2	19	99	56	72	6	13	34	16	77	56
3	37	75	67	5	46	98	57	19	14	7
4	4	57	32	78	56	12	43	61	3	88
5	96	16	92	18	50	90	35	15	36	97
6	75	4	38	53	1	79	56	73	45	56
7	15	76	11	93	87	8	2	58	86	94
8	51	14	60	57	74	42	59	71	88	52
9	49	6	43	39	17	18	95	6	44	75
	0	1	2	3	4	5	6	7	8	9

```
matrix = np.array([
    [2, 11, 18, 47, 33, 48, 9, 31, 8, 41],
    [55, 1, 8, 3, 91, 56, 17, 54, 23, 12],
    [19, 99, 56, 72, 6, 13, 34, 16, 77, 56],
    [37, 75, 67, 5, 46, 98, 57, 19, 14, 7],
    [4, 57, 32, 78, 56, 12, 43, 61, 3, 88],
    [96, 16, 92, 18, 50, 90, 35, 15, 36, 97],
    [75, 4, 38, 53, 1, 79, 56, 73, 45, 56],
    [15, 76, 11, 93, 87, 8, 2, 58, 86, 94],
    [51, 14, 60, 57, 74, 42, 59, 71, 88, 52],
    [49, 6, 43, 39, 17, 18, 95, 6, 44, 75]
])
```

### Lösung

- Rot: matrix[1,3]
- Grün: matrix[4:6,2:6]
- Pink: matrix[:,7]
- Orange: matrix[7,:5]
- Blau: matrix[-1,-1]

# 31 Array Manipulation

## 31.1 Ändern der Form

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
print(matrix)
```

```
print(np.transpose(matrix))
```

```
vector = matrix.flatten()
print(vector)
```

```
print(np.reshape(matrix, [3, 2]))
```

```
liste = np.array([1, 2, 3, 4, 5, 6])  
  
neue_liste = np.append(liste, 7)  
print(neue_liste)
```

```
liste = np.array([1, 2, 3, 4, 5, 6])  
  
neue_liste = np.insert(liste, 3, 7)  
print(neue_liste)
```

```
liste = np.array([1, 2, 3, 4, 5, 6])  
  
neue_liste = np.delete(liste, 3)  
print(neue_liste)
```

```
a = np.array([1, 2, 3, 4, 5, 6])
b = np.array([7, 8, 9, 10])

neue_liste = np.concatenate((a, b))
print(neue_liste)
```

## 31.2 Sortieren von Arrays

```
import numpy as np
unsortiert = np.array([4, 2, 1, 6, 3, 5])

sortiert = np.sort(unsortiert)

print(sortiert)
```

## 31.3 Unterlisten mit einzigartigen Werten

```
import numpy as np
liste_mit_dopplungen = np.array([4, 1, 1, 6, 3, 4, 7, 3, 3])

einzigartige_werte = np.unique(liste_mit_dopplungen)

print(einzigartige_werte)
```

```
import numpy as np
liste_mit_dopplungen = np.array([4, 1, 1, 6, 3, 4, 7, 3, 3])

einzigartige_werte, anzahl = np.unique(liste_mit_dopplungen, return_counts=True)

print(anzahl)
```

### 💡 Zwischenübung: Array-Manipulation

Gegeben ist das folgende zweidimensionale Array matrix:

```
matrix = np.array([
    [4, 7, 2, 8],
    [1, 5, 3, 6],
    [9, 2, 4, 7]
])
```

1. Ändern Sie die Form des Arrays matrix in ein eindimensionales Array.
2. Sortieren Sie das eindimensionale Array in aufsteigender Reihenfolge.
3. Ändern Sie die Form des sortierten Arrays in ein zweidimensionales Array mit 2 Zeilen und 6 Spalten.
4. Bestimmen Sie die eindeutigen Elemente im ursprünglichen Array matrix und geben Sie diese aus.

### Lösung

```
matrix = np.array([
    [4, 7, 2, 8],
    [1, 5, 3, 6],
    [9, 2, 4, 7]
])

# 1. Ändern der Form in ein eindimensionales Array
flat_array = matrix.flatten()

# 2. Sortieren des eindimensionalen Arrays in aufsteigender Reihenfolge
sorted_array = np.sort(flat_array)

# 3. Ändern der Form des sortierten Arrays in ein 2x6-Array
reshaped_array = sorted_array.reshape(2, 6)

# 4. Bestimmen der eindeutigen Elemente im ursprünglichen Array
unique_elements_original = np.unique(matrix)
```

# 32 Lesen und Schreiben von Dateien

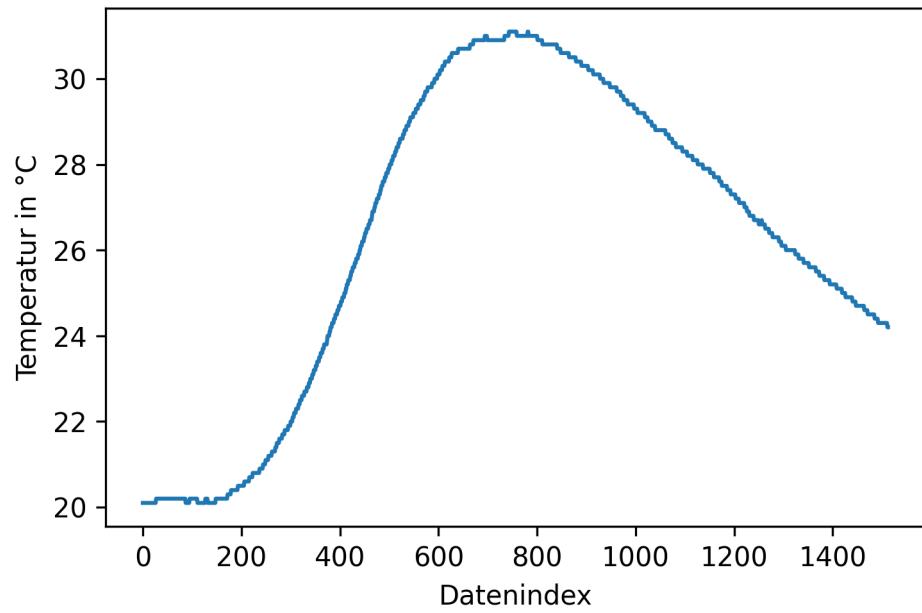
## 32.1 Lesen von Dateien

TC01.csv

```
dateiname = '01-daten/TC01.csv'  
daten = np.loadtxt(dateiname)
```

```
print("Daten:", daten)  
print("Form:", daten.shape)
```

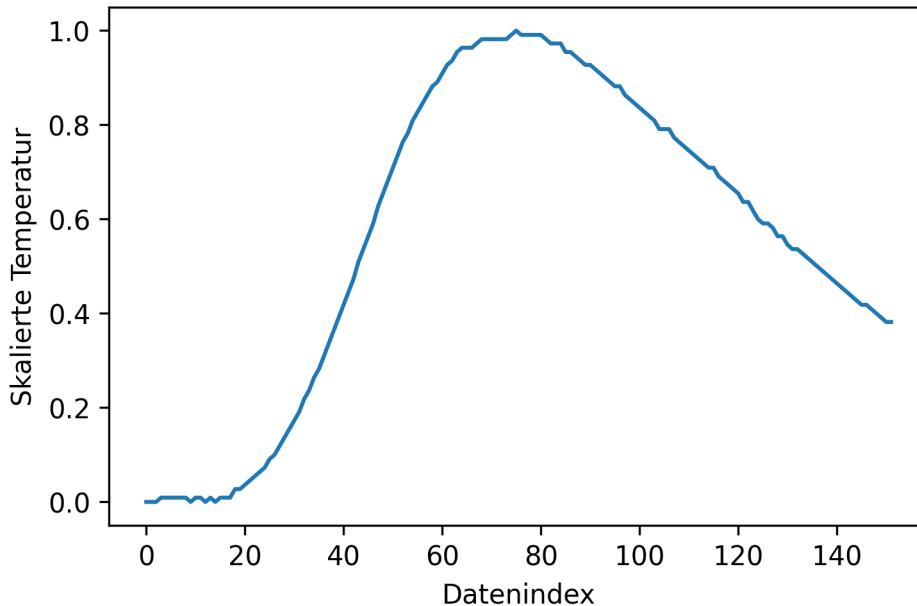
```
plt.plot(daten)  
plt.xlabel('Datenindex')  
plt.ylabel('Temperatur in °C');
```



## 32.2 Schreiben von Dateien

```
wertebereich = np.max(daten) - np.min(daten)
daten_skaliert = ( daten - np.min(daten) ) / wertebereich
daten_skaliert = daten_skaliert[::10]
```

```
plt.plot(daten_skaliert)
plt.xlabel('Datenindex')
plt.ylabel('Skalierte Temperatur');
```



```
# Zuweisung ist auf mehrere Zeilen aufgeteilt, aufgrund der
# schmalen Darstellung im Skript
kommentar = f'Daten aus {dateiname} skaliert auf den Bereich ' + \
            '0 bis 1 \noriginales Min / Max:' + \
            f'{np.min(daten)}/{np.max(daten)}'
neu_dateiname = '01-daten/TC01_skaliert.csv'

np.savetxt(neu_dateiname, daten_skaliert,
           header=kommentar, fmt='%5.2f')
```

```
# Einlesen der ersten Zeilen der neu erstellten Datei
datei = open(neu_dateiname, 'r')
for i in range(10):
    print( datei.readline() , end=' ')
datei.close()
```

## 33 Arbeiten mit Bildern

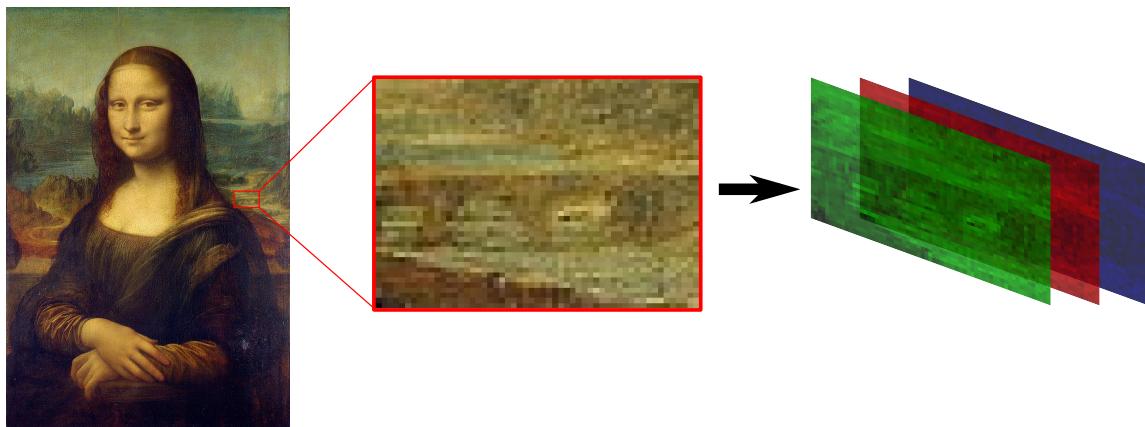


Figure 33.1: Ein hochauflößtes Bild besteht aus sehr vielen Pixeln. Jedes Pixel enthält 3 Farbwerte, einen für die Farbe Rot, einen für Grün und einen für Blau.

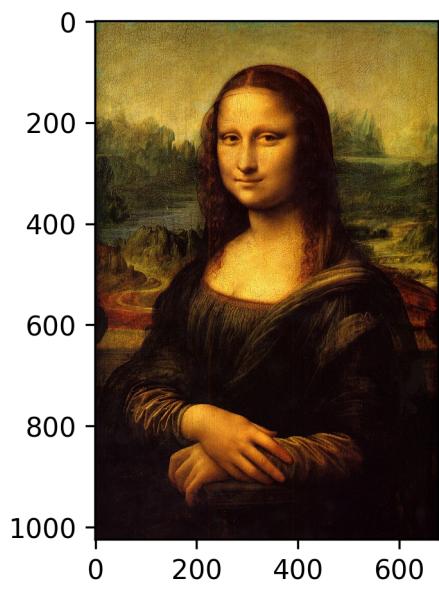
[Link](#)

```
import matplotlib.pyplot as plt

data = plt.imread("00-bilder/mona_lisa.jpg")
print("Form:", data.shape)
```

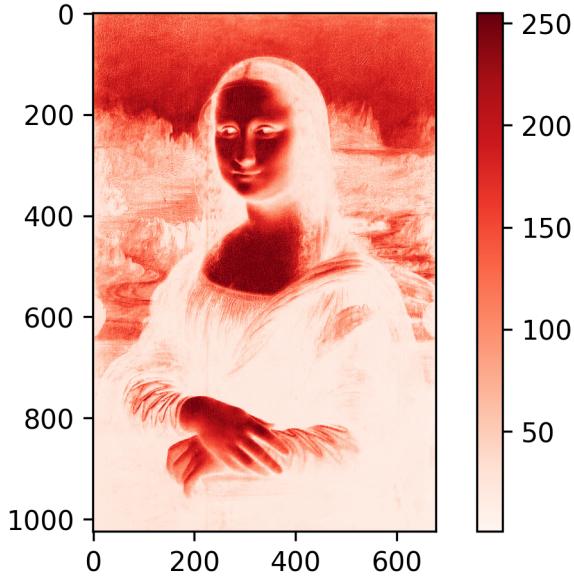
```
print(data)
```

```
plt.imshow(data)
```

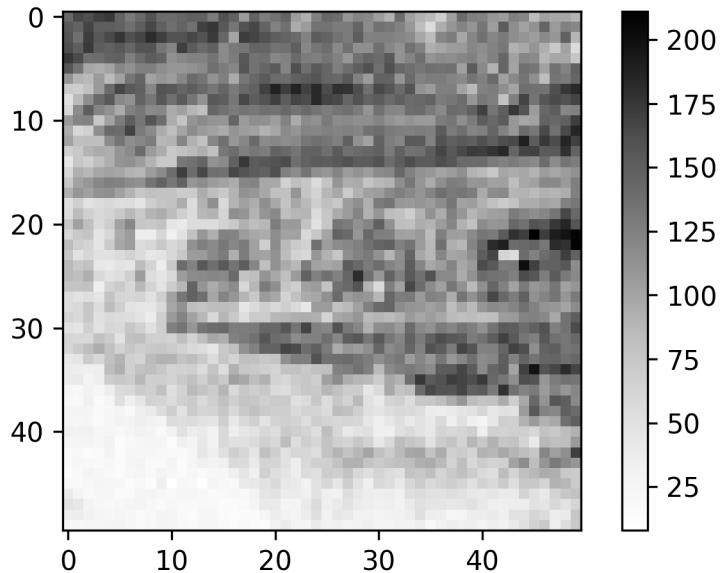


```
# Als Farbskala wird die Rotskala  
# verwendet 'Reds'
```

```
plt.imshow( data[:, :, 0], cmap='Reds' )
plt.colorbar()
plt.show()
```



```
bereich = np.array(data[450:500, 550:600, 0], dtype=float)
plt.imshow( bereich, cmap="Greys" )
plt.colorbar()
```



Laplace-Operator

```

def img_lap(data, schwellwert=25):

    # Erstellung einer Kopie der Daten, nun jedoch als
    # Array mit Gleitkommazahlen
    bereich = np.array(data, dtype=float)

    # Aufteilung der obigen Gleichung in zwei Teile
    lapx = bereich[2:, :] - 2*bereich[1:-1, :] + bereich[:-2, :]
    lapy = bereich[:, 2:] - 2*bereich[:, 1:-1] + bereich[:, :-2]

    # Zusammenführung der Teile und Bildung des Betrags
    lap = np.abs(lapx[:,1:-1] + lapy[1:-1, :])

    # Schwellwertanalyse
    lap[lap > schwellwert] = 255

```

```

lap[lap < schwellwert] = 0

return lap

```

Bild

```

data = plt.imread('01-daten/campus_haspel.jpeg')
bereich = np.array(data[1320:1620, 400:700, 1], dtype=float)

lap = img_lap(bereich)

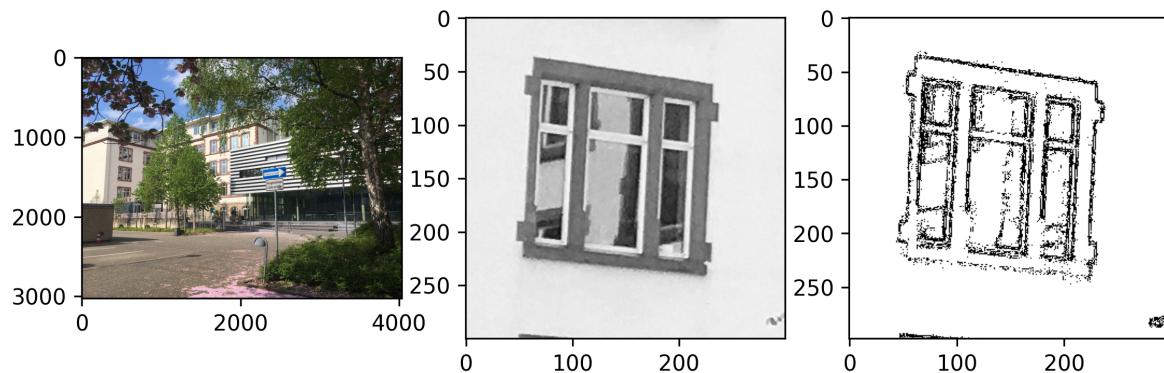
plt.figure(figsize=(9, 3))

ax = plt.subplot(1, 3, 1)
ax.imshow(data, cmap="Greys_r")

ax = plt.subplot(1, 3, 2)
ax.imshow(bereich, cmap="Greys_r");

ax = plt.subplot(1, 3, 3)
ax.imshow(lap, cmap="Greys");

```



```

data = plt.imread('01-daten/campus_haspel.jpeg')

# Speichern der einzelnen Farben in Arrays
rot = np.array(data[:, :, 0], dtype=float)
gruen = np.array(data[:, :, 1], dtype=float)
blau = np.array(data[:, :, 2], dtype=float)

# Setzen wir den Bereich des linken Baumes im Array auf 0
gruen_neu = gruen.copy()
gruen_neu[800:2000, 700:1700] = 0

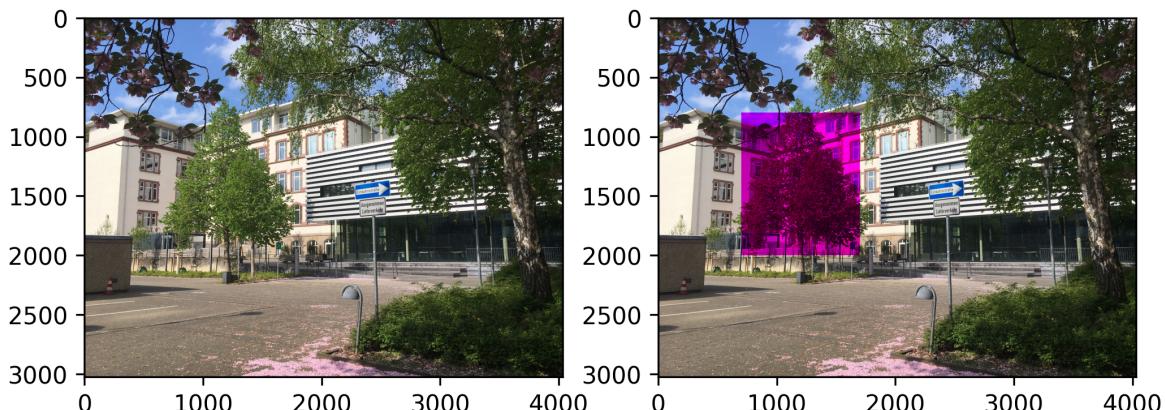
zusammengesetzt = np.dstack((rot, gruen_neu, blau)).astype('uint8')

plt.figure(figsize=(8, 5))

ax = plt.subplot(1, 2, 1)
ax.imshow(data, cmap="Greys_r")

ax = plt.subplot(1, 2, 2)
ax.imshow(zusammengesetzt)

```



💡 Zwischenübung: Bilder bearbeiten

Lesen Sie folgendes Bild vom Haspel Campus in Wuppertal ein: [Bild](#).  
Extrahieren Sie den blauen Anteil und lassen Sie sich die Zeile in der Mitte des Bildes sowie einen beliebigen Bildausschnitt ausgeben.

## Lösung

```
import numpy as np
import matplotlib.pyplot as plt

data = plt.imread('01-daten/campus_haspel.jpeg')

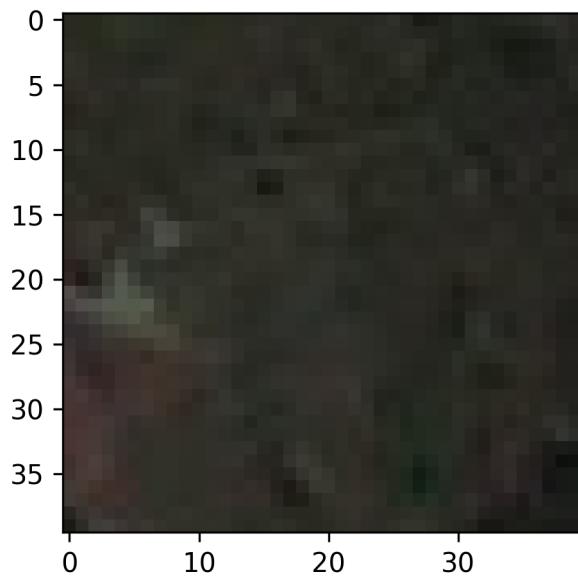
form = data.shape
print("Form:", data.shape)

blau = data[:, :, 2]
plt.imshow(blau, cmap='Blues')

zeile = data[int(form[0]/2), :, 2]
print(zeile)

ausschnitt = data[10:50, 10:50, :]
plt.imshow(ausschnitt)
```

Form: (3024, 4032, 3)  
[221 220 220 ... 28 28 28]



## **34 Lernzielkontrolle**

### **Aufgabe 1**

### **Aufgabe 2**

- 1.
- 2.
- 3.
- 4.

## Aufgabe 3

## Aufgabe 4

```
vector = np.array([ 4.8,  8.2, 15.6, 16.6, 23.2, 42.8 ])
```

## Aufgabe 5

- 1.
- 2.
- 3.

## Aufgabe 6

- 1.
- 2.

## Aufgabe 7

```
matrix = np.array([
    [ 1,  2,  3,  4,  5],
    [ 6,  7,  8,  9, 10],
    [11, 12, 13, 14, 15],
    [16, 17, 18, 19, 20],
```

```
[21, 22, 23, 24, 25]  
])
```

- 1.
- 2.
- 3.

## Aufgabe 8

```
array = np.arange(1, 21)
```

- 1.
- 2.
- 3.
- 4.
- 5.

## Aufgabe 9

## Aufgabe 10

Lösung

### Aufgabe 1

```
import numpy as np
```

## Aufgabe 2

```
# 1.  
np.array([1, 2, 3])  
  
# 2.  
print(np.arange(10))  
  
# 3.  
print(np.ones((3, 3)))  
  
# 4.  
print(np.arange(10, 51, 5))  
  
[0 1 2 3 4 5 6 7 8 9]  
[[1. 1. 1.]  
 [1. 1. 1.]  
 [1. 1. 1.]]  
[10 15 20 25 30 35 40 45 50]
```

## Aufgabe 3

`np.ndim()`: Gibt die Anzahl der Dimensionen zurück. `np.shape()`: Gibt die Längen der einzelnen Dimensionen wieder. `np.size()`: Gibt die Anzahl aller Elemente aus.

## Aufgabe 4

Da es sich hier um Gleitkommazahlen handelt, ist der Datentyp `float64`.

```
vector = np.array([ 4.8,  8.2, 15.6, 16.6, 23.2, 42.8 ])  
print(vector.dtype)  
  
float64
```

## Aufgabe 5

```

a = np.array([5, 1, 3, 6, 4])
b = np.array([6, 5, 2, 6, 9])

# 1.
ergebnis = a + b
print("Die Summe beider Vektoren ergibt: ", ergebnis)

# 2.
ergebnis = a * b
print("Das Produkt beider Vektoren ergibt: ", ergebnis)

# 3.
ergebnis = a + 3
print("Die Summe von a und 3 ergibt: ", ergebnis)

```

Die Summe beider Vektoren ergibt: [11 6 5 12 13]  
 Das Produkt beider Vektoren ergibt: [30 5 6 36 36]  
 Die Summe von a und 3 ergibt: [8 4 6 9 7]

## Aufgabe 6

```

a = np.array([9, 2, 3, 1, 3])

# 1.
mittelwert = np.mean(a)
print("Der Mittelwert ist: ", mittelwert)

standardabweichung = np.std(a)
print("Die Standardabweichung von a beträgt: ", standardabweichung)

# 2.
minimum = np.min(a)
print("Das Minimum beträgt: ", minimum)

maximum = np.max(a)
print("Das Maximum beträgt: ", maximum)

```

Der Mittelwert ist: 3.6  
 Die Standardabweichung von a beträgt: 2.8000000000000003  
 Das Minimum beträgt: 1  
 Das Maximum beträgt: 9

## Aufgabe 7

```
matrix = np.array([
    [ 1,  2,  3,  4,  5],
    [ 6,  7,  8,  9, 10],
    [11, 12, 13, 14, 15],
    [16, 17, 18, 19, 20],
    [21, 22, 23, 24, 25]
])

# 1. Erste Zeile
print(matrix[0,:])

# 2. Letzte Spalte
print(matrix[:, -1])

# 3. Ausschnitt
print(matrix[1:4, 0:3])
```

```
[1 2 3 4 5]
[ 5 10 15 20 25]
[[ 6  7  8]
 [11 12 13]
 [16 17 18]]
```

## Aufgabe 8

```

array = np.arange(1, 21)

# 1. Ändern der Form in eine 4x5-Matrix
matrix_4x5 = array.reshape(4, 5)

# 2. Ändern der Form in eine 5x4-Matrix
matrix_5x4 = array.reshape(5, 4)

# 3. Ändern der Form in eine 2x2x5-Matrix
matrix_2x2x5 = array.reshape(2, 2, 5)

# 4. Abflachen der 2x2x5-Matrix zu einem eindimensionalen Array
flattened_array = matrix_2x2x5.flatten()

# 5. Transponieren der 4x5-Matrix
transposed_matrix = matrix_4x5.T

# Ausgabe der Ergebnisse (optional)
print("Originales Array:", array)
print("4x5-Matrix:\n", matrix_4x5)
print("5x4-Matrix:\n", matrix_5x4)
print("2x2x5-Matrix:\n", matrix_2x2x5)
print("Abgeflachtes Array:", flattened_array)
print("Transponierte 4x5-Matrix:\n", transposed_matrix)

```

```

Originales Array: [ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20]
4x5-Matrix:
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]
 [11 12 13 14 15]
 [16 17 18 19 20]]
5x4-Matrix:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]]
2x2x5-Matrix:
[[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
```

```
[[11 12 13 14 15]
 [16 17 18 19 20]]
```

Abgeflachtes Array: [ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]

Transponierte 4x5-Matrix:

```
[[ 1 6 11 16]
 [ 2 7 12 17]
 [ 3 8 13 18]
 [ 4 9 14 19]
 [ 5 10 15 20]]
```

### Aufgabe 9

Die passenden Funktionen sind `np.loadtxt()` und `np.savetxt()`.

### Aufgabe 10

Typischerweise sind Bilddaten große Matrizen, wobei die Farben in drei unterschiedlichen Matrizen gespeichert werden. Dabei ist die Farbreihenfolge oft “Rot”, “Grün” und “Blau”. Dementsprechen müssen wir, wenn die Daten in der Matrix `data` gespeichert sind, mit Slicing eine Dimension auswählen: `data[:, :, 0]`, wobei die Zahl 0-2 für die jeweilige Farbe steht.

# 35 Übung

## 35.1 Aufgabe 1 Filmdatenbank

- 1.
- 2.
- 3.

```
import numpy as np

bewertungen = np.array([
    [1, 101, 4.5],
    [1, 102, 3.0],
    [2, 101, 2.5],
    [2, 103, 4.0],
    [3, 101, 5.0],
    [3, 104, 3.5],
    [3, 105, 4.0]
])
```

- 💡 a) Bestimmen Sie die niedrigste und höchste Bewertung, die jemals gegeben wurde

### Lösung

```
niedrigste_bewertung = np.min(bewertungen[:,2])

print("Die niedrigste jemals gegebene Bewertung ist:", niedrigste_bewertung)

hoechste_bewertung = np.max(bewertungen[:,2])

print("Die höchste jemals gegebene Bewertung ist:", hoechste_bewertung)
```

Die niedrigste jemals gegebene Bewertung ist: 2.5  
Die höchste jemals gegebene Bewertung ist: 5.0

💡 b) Nennen Sie alle Bewertungen für Film 1

**Lösung**

```
bewertungen_film_1 = bewertungen[np.where(bewertungen[:,0]==1)]  
  
print("Bewertungen für Film 1:\n", bewertungen_film_1)
```

Bewertungen für Film 1:  
[[ 1. 101. 4.5]  
 [ 1. 102. 3. ]]

💡 c) Nennen Sie alle Bewertungen von Person 101

**Lösung**

```
bewertungen_101 = bewertungen[np.where(bewertungen[:,1]==101)]  
  
print("Bewertungen von Person 101:\n", bewertungen_101)
```

Bewertungen von Person 101:  
[[ 1. 101. 4.5]  
 [ 2. 101. 2.5]  
 [ 3. 101. 5. ]]

💡 d) Berechnen Sie die mittlere Bewertung für jeden Film und geben Sie diese nacheinander aus

**Lösung**

```
for ID in [1, 2, 3]:  
  
    mittelwert = np.mean(bewertungen[np.where(bewertungen[:,0]==ID),2])  
  
    print("Die mittlere Bewertung für Film", ID, "beträgt:", mittelwert)
```

Die mittlere Bewertung für Film 1 beträgt: 3.75

Die mittlere Bewertung für Film 2 beträgt: 3.25  
Die mittlere Bewertung für Film 3 beträgt: 4.1666666666666667

- 💡 e) Finden Sie den Film mit der höchsten Bewertung

**Lösung**

```
index_hoehste_bewertung = np.argmax(bewertungen[:,2])  
  
print(bewertungen[index_hoehste_bewertung,:])
```

[ 3. 101. 5.]

- 💡 f) Finden Sie die Person mit den meisten Bewertungen

**Lösung**

```
einzigartige_person, anzahl = np.unique(bewertungen[:, 1], return_counts=True)  
  
meist_aktive_person = einzigartige_person[np.argmax(anzahl)]  
  
print("Personen mit den meisten Bewertungen:", meist_aktive_person)
```

Personen mit den meisten Bewertungen: 101.0

- 💡 g) Nennen Sie alle Filme mit einer Wertung von 4 oder besser.

**Lösung**

```
index_bewertung_besser_vier = bewertungen[:,2] >= 4  
  
print("Filme mit einer Wertung von 4 oder besser:")  
  
print(bewertungen[index_bewertung_besser_vier,:])
```

Filme mit einer Wertung von 4 oder besser:  
[[ 1. 101. 4.5]  
 [ 2. 103. 4. ]  
 [ 3. 101. 5. ]  
 [ 3. 105. 4. ]]

💡 h) Film Nr. 4 ist erschienen. Der Film wurde von Person 102 mit einer Note von 3.5 bewertet. Fügen Sie diesen zur Datenbank hinzu.

### Lösung

```
neue_bewertung = np.array([4, 102, 3.5])

bewertungen = np.append(bewertungen, [neue_bewertung], axis=0)

print(bewertungen)
```

```
[[ 1. 101. 4.5]
 [ 1. 102. 3. ]
 [ 2. 101. 2.5]
 [ 2. 103. 4. ]
 [ 3. 101. 5. ]
 [ 3. 104. 3.5]
 [ 3. 105. 4. ]
 [ 4. 102. 3.5]]
```

💡 i) Person 102 hat sich Film Nr. 1 nochmal angesehen und hat das Ende jetzt doch verstanden. Dementsprechend soll die Bewertung jetzt auf 5.0 geändert werden.

### Lösung

```
bewertungen[(bewertungen[:, 0] == 1) &
              (bewertungen[:, 1] == 102), 2] = 5.0

print("Aktualisieren der Bewertung:\n", bewertungen)
```

Aktualisieren der Bewertung:

```
[[ 1. 101. 4.5]
 [ 1. 102. 5. ]
 [ 2. 101. 2.5]
 [ 2. 103. 4. ]
 [ 3. 101. 5. ]
 [ 3. 104. 3.5]
 [ 3. 105. 4. ]
 [ 4. 102. 3.5]]
```

## 35.2 Aufgabe 2 - Kryptographie - Caesar-Chiffre

Table 35.1: Ascii-Tabelle

```
import numpy as np

# Funktion, die einen Buchstaben in ihren ASCII-Wert umwandelt
def buchstabe_zu_ascii(c):
    return np.array([ord(c)])

# Funktion, die einen ASCII-Wert in den passenden Buchstaben umwandelt
```

```
def ascii_zu_buchstabe(a):
    return chr(a)
```

- 💡 1. Überlegen Sie sich zunächst, wie man diese zyklische Verschiebung mathematisch ausdrücken könnte (Hinweis: Modulo Rechnung)

**Lösung**

$$\text{ASCII}_{\text{verschoben}} = (\text{ASCII} - 97 + \text{Versatz}) \bmod 26 + 97$$

- 💡 2. Schreiben Sie Code, der mit einer Schleife alle Zeichen umwandelt.

Zunächst sollen alle Zeichen in ASCII-Code umgewandelt werden. Dann wird die Formel auf die Zahlenwerte angewendet und schlussendlich in einer dritten Schleife wieder alle Werte in Buchstaben übersetzt.

**Lösung**

```

import numpy as np

# Funktion, die einen Buchstaben in ihren ASCII-Wert umwandelt
def buchstabe_zu_ascii(c):
    return ord(c)

# Funktion, die einen ASCII-Wert in den passenden Buchstaben umwandelt
def ascii_zu_buchstabe(a):
    return chr(a)

klartext = "abracadabra"
versatz = 3

umgewandelter_text = []
verschluesselte_zahl = []
verschluesselter_text= []

for buchstabe in klartext:
    umgewandelter_text.append(buchstabe_zu_ascii(buchstabe))
print(umgewandelter_text)

for zahl in umgewandelter_text:
    verschluesselt = (zahl - 97 + versatz) % 26 + 97
    verschluesselte_zahl.append(verschluesselt)
print(verschluesselte_zahl)

for zahl in verschluesselte_zahl:
    verschluesselter_text.append(ascii_zu_buchstabe(zahl))
print(verschluesselter_text)

```

```

[97, 98, 114, 97, 107, 97, 100, 97, 98, 114, 97]
[100, 101, 117, 100, 110, 100, 103, 100, 101, 117, 100]
['d', 'e', 'u', 'd', 'n', 'd', 'g', 'd', 'e', 'u', 'd']

```

- 💡 3. Ersetzen Sie die Schleife, indem Sie die Rechenoperation mit einem NumPy-Array durchführen

### Lösung

```
import numpy as np

# Funktion, die einen Buchstaben in ihren ASCII-Wert umwandelt
def buchstabe_zu_ascii(c):
    return ord(c)

# Funktion, die einen ASCII-Wert in den passenden Buchstaben umwandelt
def ascii_zu_buchstabe(a):
    return chr(a)

klartext = "abracadabra"
versatz = 3

umgewandelter_text = []
verschlüsselte_zahl = []
verschlüsselter_text= []

for buchstabe in klartext:
    umgewandelter_text.append(buchstabe_zu_ascii(buchstabe))
print(umgewandelter_text)

umgewandelter_text = np.array(umgewandelter_text)
verschlüsselte_zahl = (umgewandelter_text - 97 + versatz) % 26 + 97
print(verschlüsselte_zahl)

for zahl in verschlüsselte_zahl:
    verschlüsselter_text.append(ascii_zu_buchstabe(zahl))
print(verschlüsselter_text)

[97, 98, 114, 97, 107, 97, 100, 97, 98, 114, 97]
[100 101 117 100 110 100 103 100 101 117 100]
['d', 'e', 'u', 'd', 'n', 'd', 'g', 'd', 'e', 'u', 'd']
```

- 💡 4. Schreiben sie den Code so um, dass der verschlüsselte Text entschlüsselt wird.

### Lösung

```
import numpy as np

# Funktion, die einen Buchstaben in ihren ASCII-Wert umwandelt
def buchstabe_zu_ascii(c):
    return ord(c)

# Funktion, die einen ASCII-Wert in den passenden Buchstaben umwandelt
def ascii_zu_buchstabe(a):
    return chr(a)

versatz = 3

umgewandelter_text = []
verschluesselte_zahl = []
entschluesselter_text= []

for buchstabe in verschluesselter_text:
    umgewandelter_text.append(buchstabe_zu_ascii(buchstabe))
print(umgewandelter_text)

umgewandelter_text = np.array(umgewandelter_text)
verschluesselte_zahl = (umgewandelter_text - 97 - versatz) % 26 + 97
print(verschluesselte_zahl)

for zahl in verschluesselte_zahl:
    entschluesselter_text.append(ascii_zu_buchstabe(zahl))
print(entschluesselter_text)

[100, 101, 117, 100, 110, 100, 103, 100, 101, 117, 100]
[ 97  98 114  97 107  97 100  97  98 114  97]
['a', 'b', 'r', 'a', 'k', 'a', 'd', 'a', 'b', 'r', 'a']
```

# 36 Klausurfragen

## Aufgabe 1

1.

2.

3.

4.

5.

## **Part V**

# **w-python-matplotlib**

# Preamble



Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch und Sebastian Seipel. “Werkzeugbaustein Matplotlib” von Marc Fehr ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar unter <https://github.com/bausteine-der-datenanalyse/w-python-matplotlib>. Ausgenommen von der Lizenz sind alle Logos und anders gekennzeichneten Inhalte. 2025

<https://github.com/bausteine-der-datenanalyse/w-python-matplotlib>

# **Intro**

## **Voraussetzungen**

- 
- 

## **Verwendete Pakete und Datensätze**

- 

## **Bearbeitungszeit**

## **Lernziele**

- 
- 
-

# 37 Einführung in Matplotlib

## 37.1 Warum Matplotlib?

- 
- 
- 
- 

## 37.2 Alternativen zu Matplotlib

## 37.3 Erstes Beispiel: Einfache Linie plotten

```
import matplotlib.pyplot as plt
import numpy as np

# Beispiel-Daten
t = np.linspace(0, 10, 100)
y = np.sin(t)

# Erstellen des Plots
plt.plot(t, y, label='sin(t)')
plt.xlabel('Zeit (s)')
```

```
plt.ylabel('Amplitude')
plt.title('Einfaches Linien-Diagramm')
plt.legend()
plt.show()
```

## 37.4 Nächste Schritte

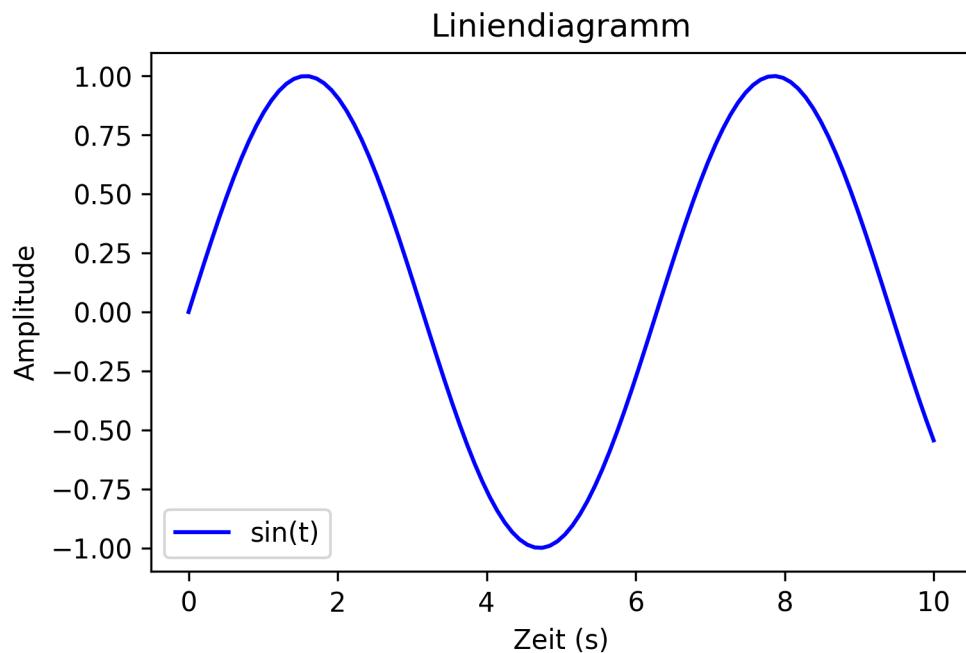
# 38 Diagrammtypen in Matplotlib

## 38.1 1. Liniendiagramme (plt.plot())

```
import matplotlib.pyplot as plt
import numpy as np

t = np.linspace(0, 10, 100)
y = np.sin(t)

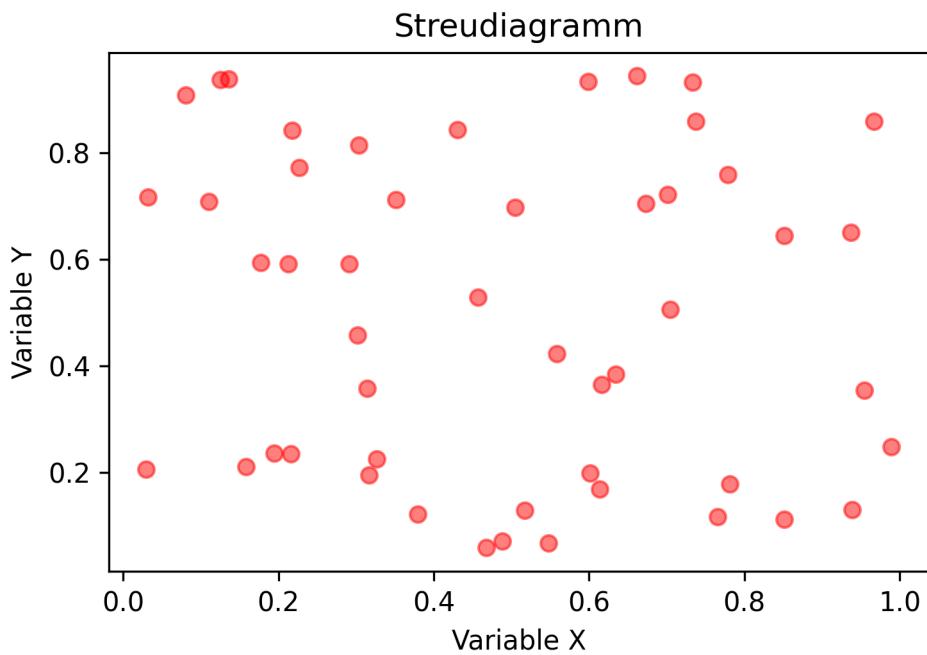
plt.plot(t, y, label='sin(t)', color='b')
plt.xlabel('Zeit (s)')
plt.ylabel('Amplitude')
plt.title('Liniendiagramm')
plt.legend()
plt.show()
```



## 38.2 2. Streudiagramme (plt.scatter())

```
x = np.random.rand(50)
y = np.random.rand(50)

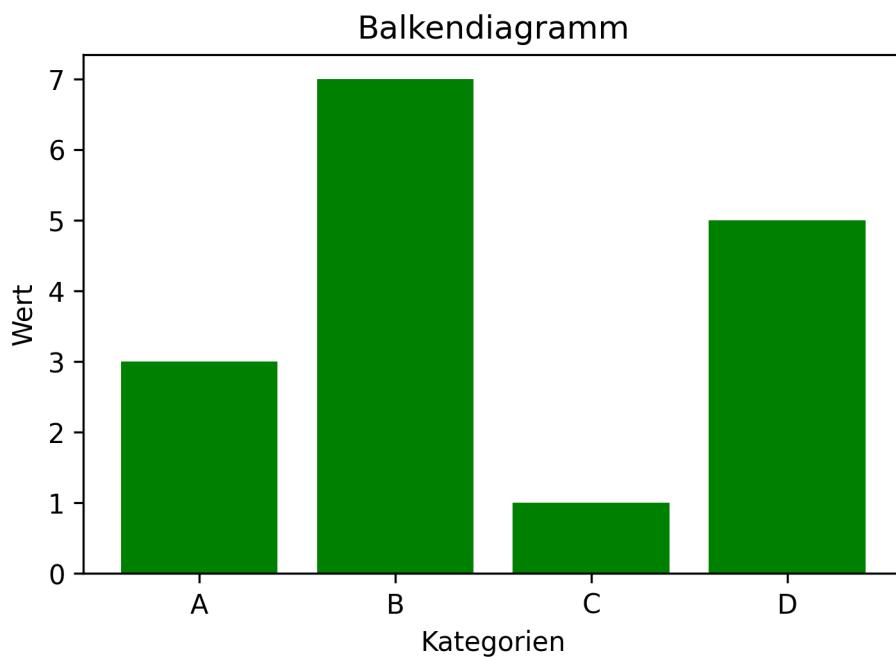
plt.scatter(x, y, color='r', alpha=0.5)
plt.xlabel('Variable X')
plt.ylabel('Variable Y')
plt.title('Streudiagramm')
plt.show()
```



### 38.3 3. Balkendiagramme (plt.bar())

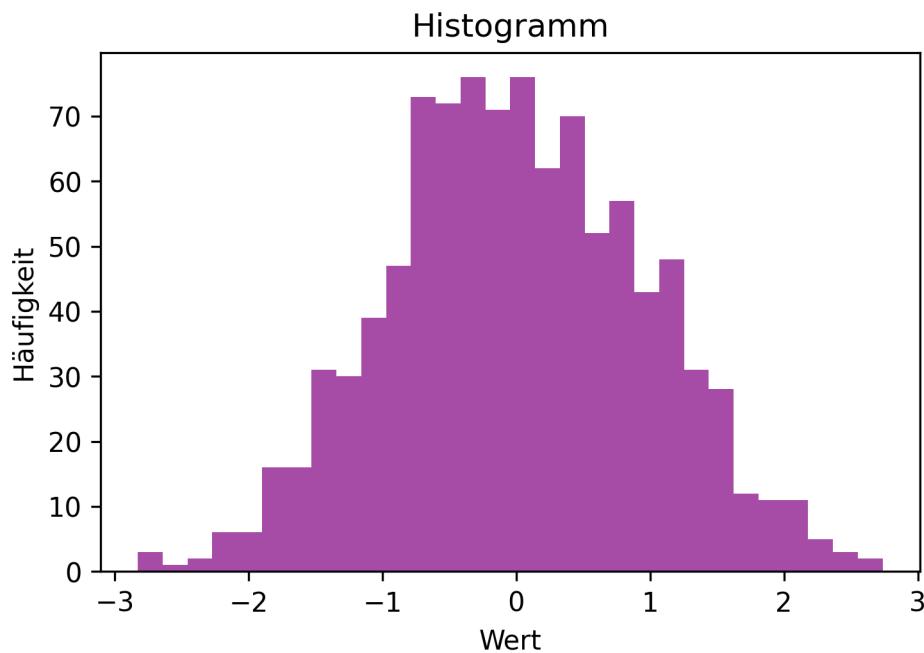
```
kategorien = ['A', 'B', 'C', 'D']
werte = [3, 7, 1, 5]

plt.bar(kategorien, werte, color='g')
plt.xlabel('Kategorien')
plt.ylabel('Wert')
plt.title('Balkendiagramm')
plt.show()
```



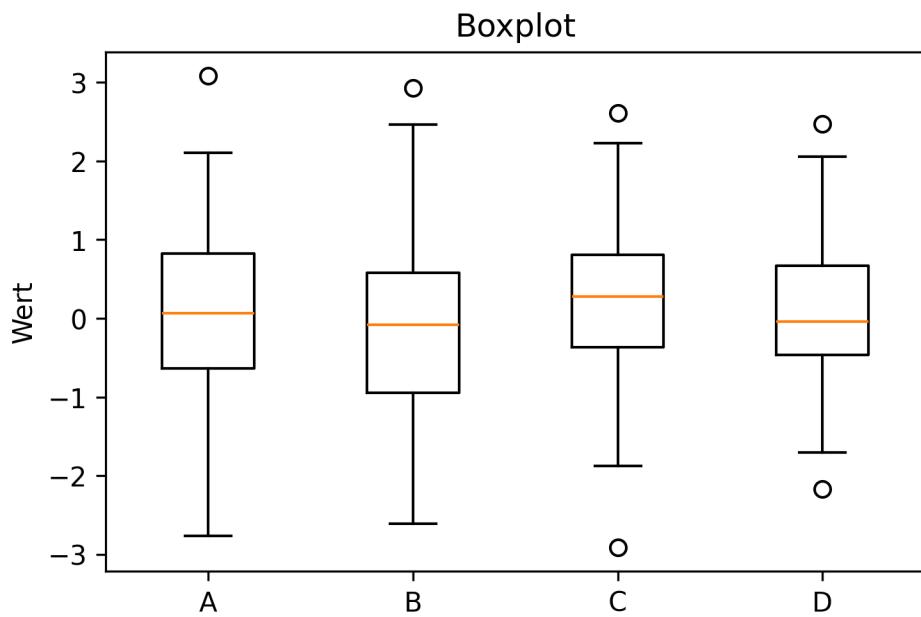
### 38.4 4. Histogramme (plt.hist())

```
daten = np.random.randn(1000)
plt.hist(daten, bins=30, color='purple', alpha=0.7)
plt.xlabel('Wert')
plt.ylabel('Häufigkeit')
plt.title('Histogramm')
plt.show()
```



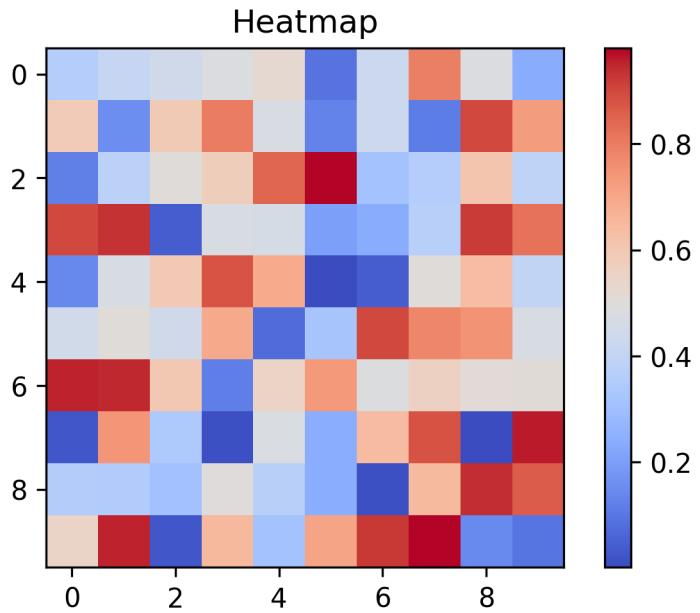
### 38.5 5. Boxplots (plt.boxplot())

```
daten = [np.random.randn(100) for _ in range(4)]
plt.boxplot(daten, labels=['A', 'B', 'C', 'D'])
plt.ylabel('Wert')
plt.title('Boxplot')
plt.show()
```



### 38.6 6. Heatmaps (plt.imshow())

```
daten = np.random.rand(10, 10)
plt.imshow(daten, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.title('Heatmap')
plt.show()
```



### 38.7 Fazit

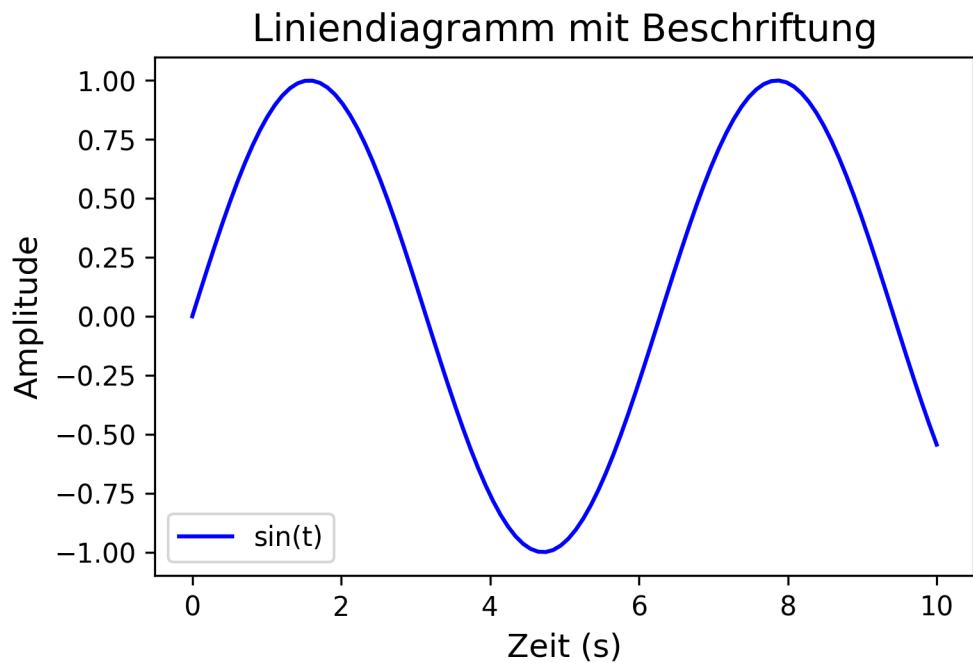
# 39 Anpassung und Gestaltung von Plots in Matplotlib

## 39.1 1. Achsentitel und Diagrammtitel

```
import matplotlib.pyplot as plt
import numpy as np

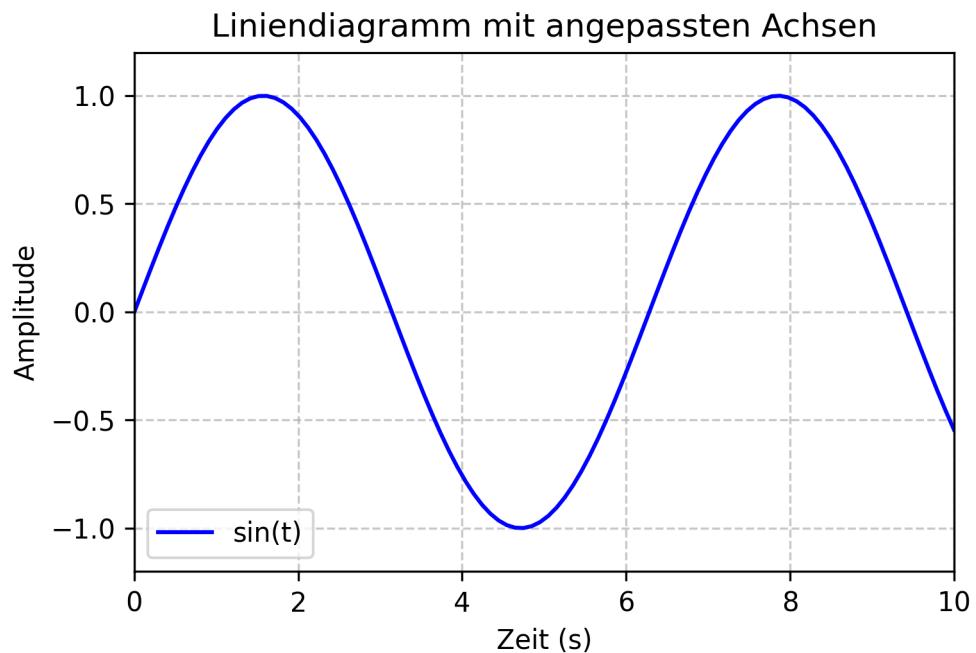
t = np.linspace(0, 10, 100)
y = np.sin(t)

plt.plot(t, y, label='sin(t)', color='b')
plt.xlabel('Zeit (s)', fontsize=12)
plt.ylabel('Amplitude', fontsize=12)
plt.title('Liniendiagramm mit Beschriftung', fontsize=14)
plt.legend()
plt.show()
```



## 39.2 2. Anpassung der Achsen

```
plt.plot(t, y, label='sin(t)', color='b')
plt.xlabel('Zeit (s)')
plt.ylabel('Amplitude')
plt.xlim(0, 10)
plt.ylim(-1.2, 1.2)
plt.grid(True, linestyle='--', alpha=0.7)
plt.title('Liniendiagramm mit angepassten Achsen')
plt.legend()
plt.show()
```

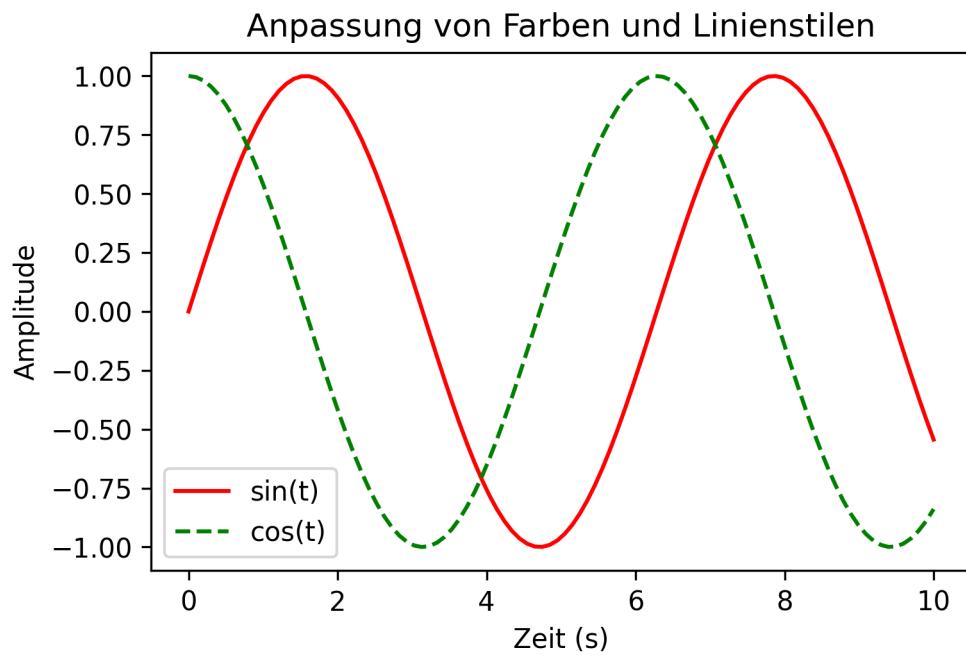


### 39.3 3. Farben und Linienstile

#### 39.3.1 Wichtige Farben (Standardfarben in Matplotlib)

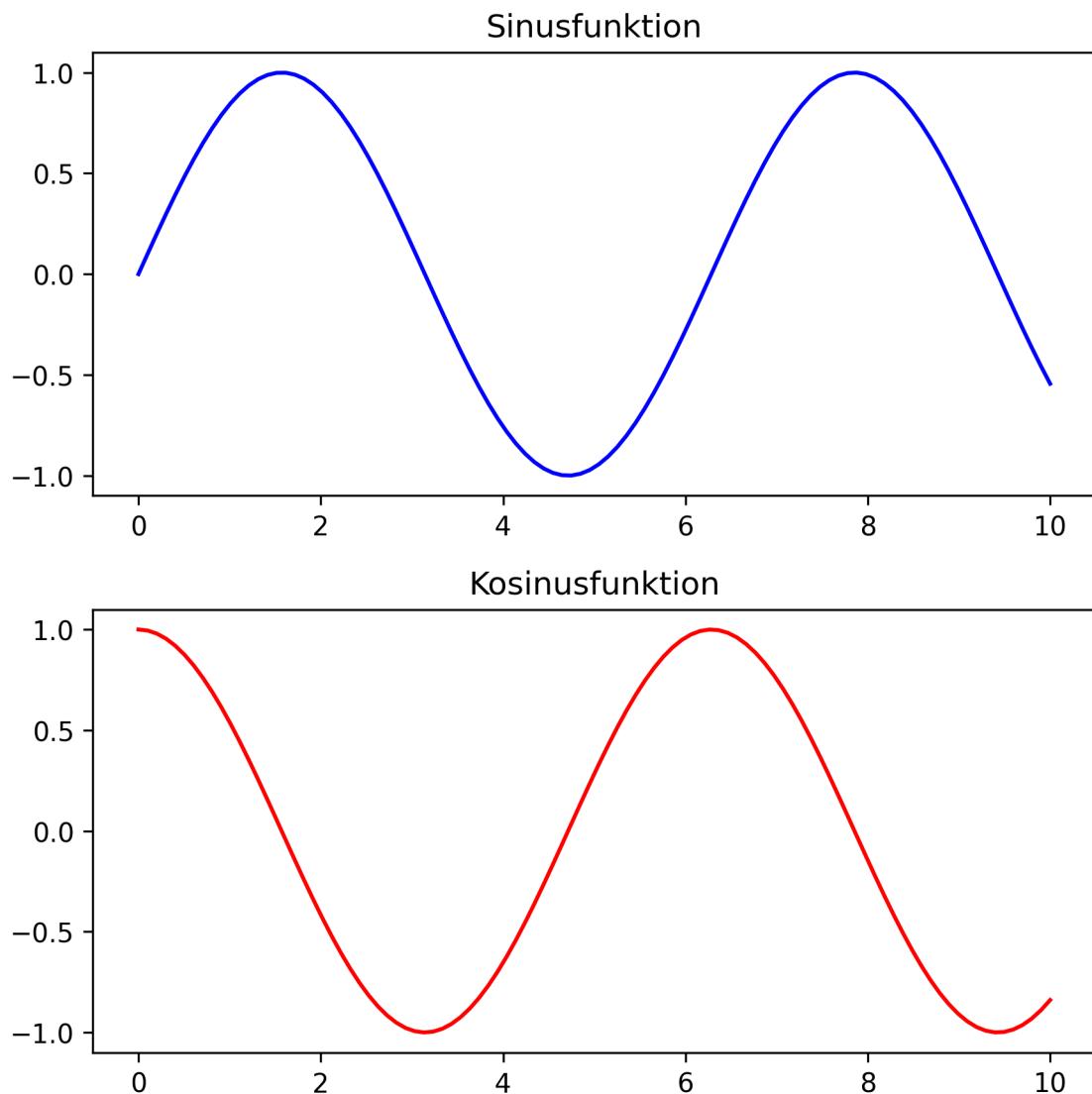
#### 39.3.2 Wichtige Linienstile

```
plt.plot(t, np.sin(t), linestyle='-', color='r', label='sin(t)')
plt.plot(t, np.cos(t), linestyle='--', color='g', label='cos(t)')
plt.xlabel('Zeit (s)')
plt.ylabel('Amplitude')
plt.title('Anpassung von Farben und Linienstilen')
plt.legend()
plt.show()
```



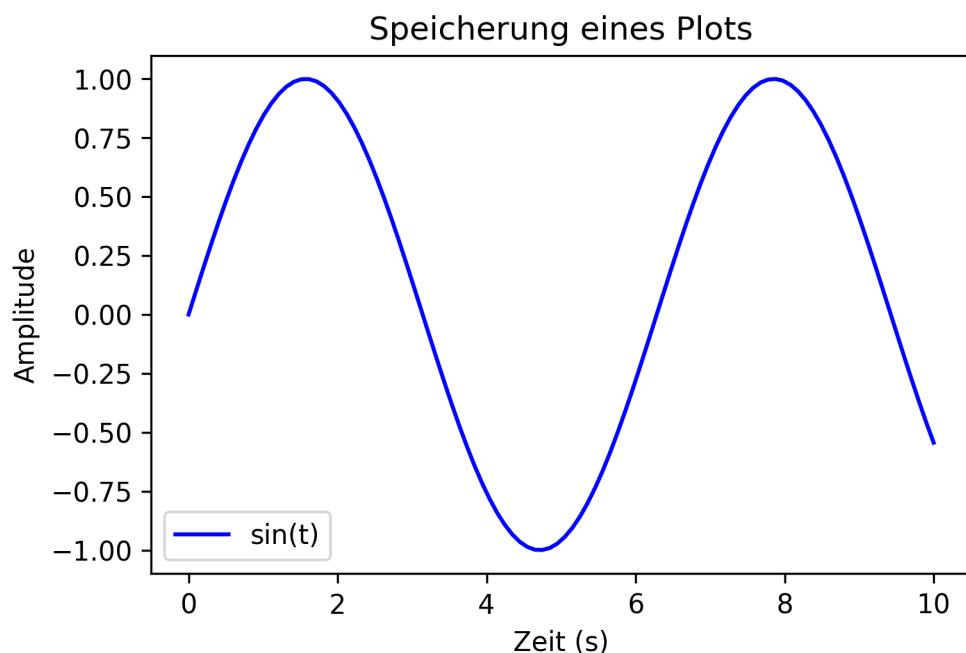
### 39.4 4. Mehrere Plots mit Subplots

```
fig, axs = plt.subplots(2, 1, figsize=(6, 6))
axs[0].plot(t, np.sin(t), color='b')
axs[0].set_title('Sinusfunktion')
axs[1].plot(t, np.cos(t), color='r')
axs[1].set_title('Kosinusfunktion')
plt.tight_layout()
plt.show()
```



## 39.5 5. Speichern von Plots

```
plt.plot(t, y, label='sin(t)', color='b')
plt.xlabel('Zeit (s)')
plt.ylabel('Amplitude')
plt.title('Speicherung eines Plots')
plt.legend()
plt.savefig('mein_plot.png', dpi=300)
plt.show()
```



## 39.6 Fazit

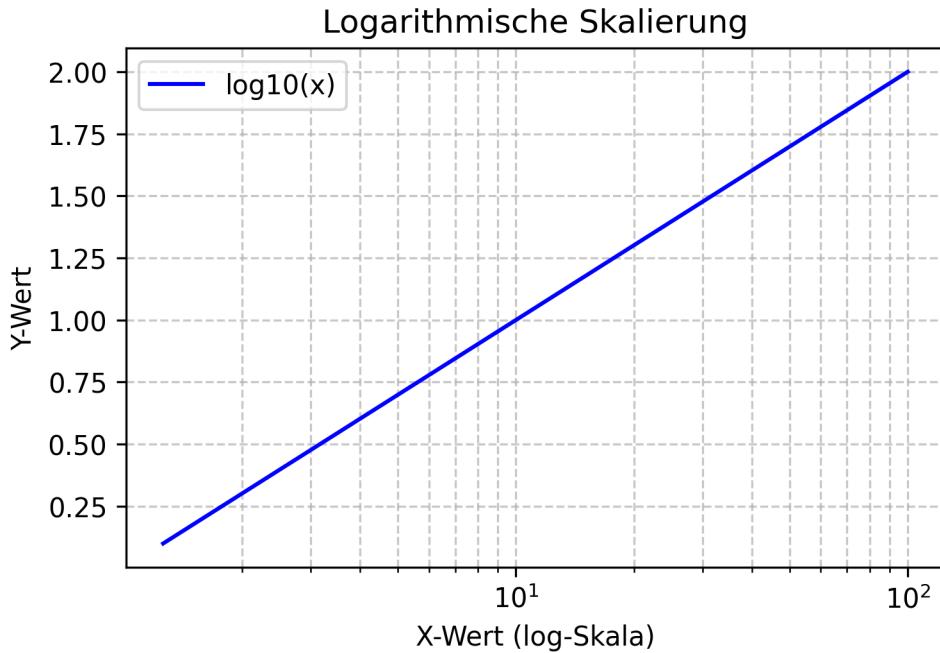
# 40 Erweiterte Techniken in Matplotlib

## 40.1 1. Logarithmische Skalen

```
import matplotlib.pyplot as plt
import numpy as np

x = np.logspace(0.1, 2, 100)
y = np.log10(x)

plt.plot(x, y, label='log10(x)', color='b')
plt.xscale('log')
plt.xlabel('X-Wert (log-Skala)')
plt.ylabel('Y-Wert')
plt.title('Logarithmische Skalierung')
plt.legend()
plt.grid(True, which='both', linestyle='--', alpha=0.7)
plt.show()
```



## 40.2 2. Twin-Achsen für verschiedene Skalierungen

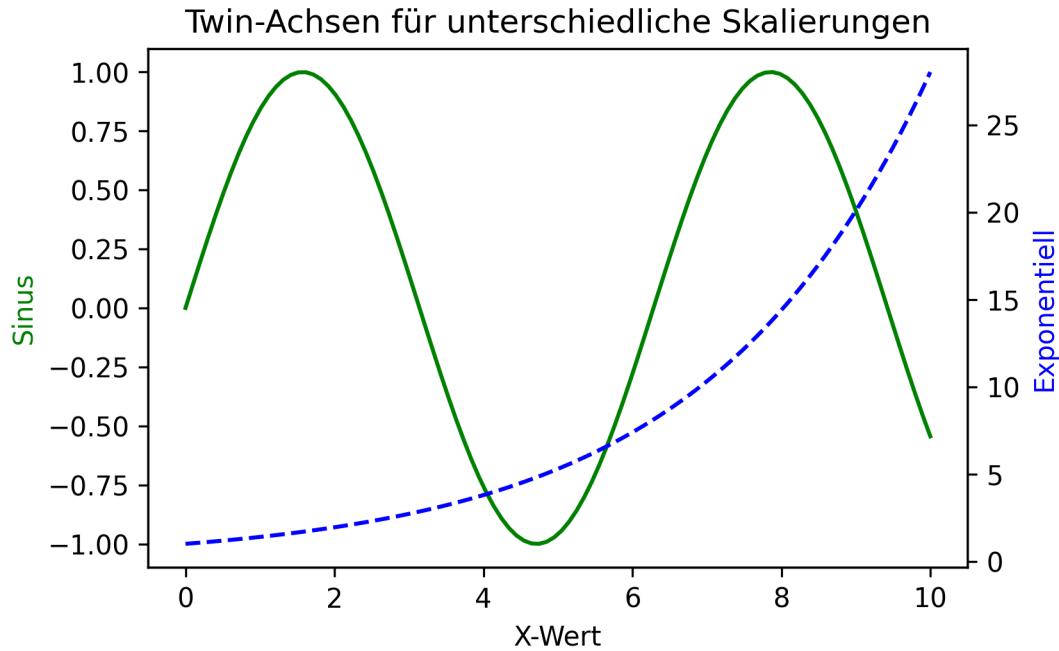
```

x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.exp(x / 3)

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(x, y1, 'g-', label='sin(x)')
ax2.plot(x, y2, 'b--', label='exp(x/3)')

ax1.set_xlabel('X-Wert')
ax1.set_ylabel('Sinus', color='g')
ax2.set_ylabel('Exponentiell', color='b')
ax1.set_title('Twin-Achsen für unterschiedliche Skalierungen')
plt.show()

```



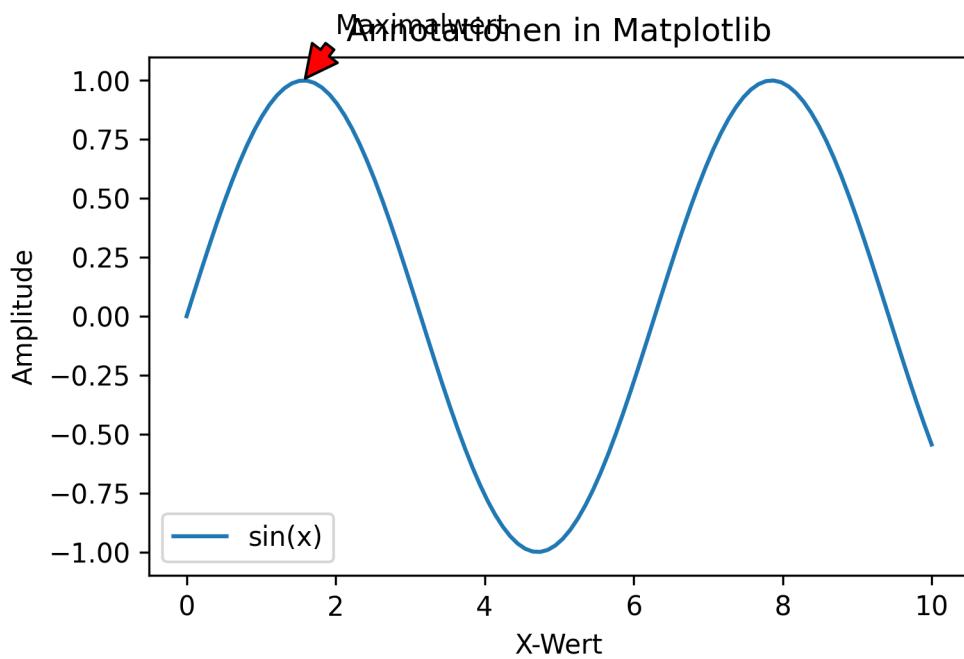
### 40.3 3. Annotationen in Diagrammen

```

x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y, label='sin(x)')
plt.xlabel('X-Wert')
plt.ylabel('Amplitude')
plt.title('Annotationen in Matplotlib')
plt.annotate('Maximalwert', xy=(np.pi/2, 1), xytext=(2, 1.2),
            arrowprops=dict(facecolor='red', shrink=0.05))
plt.legend()
plt.show()

```



## 40.4 Fazit

# 41 Best Practices in Matplotlib: Fehler und Verbesserungen

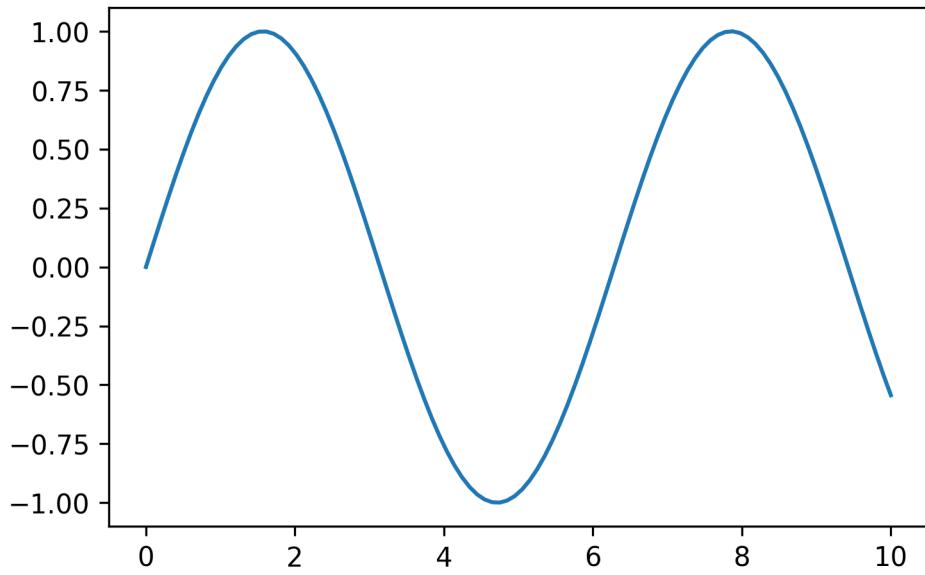
## 41.1 1. Fehlende Beschriftungen

### 41.1.1 Schlechtes Beispiel

```
import matplotlib.pyplot as plt
import numpy as np

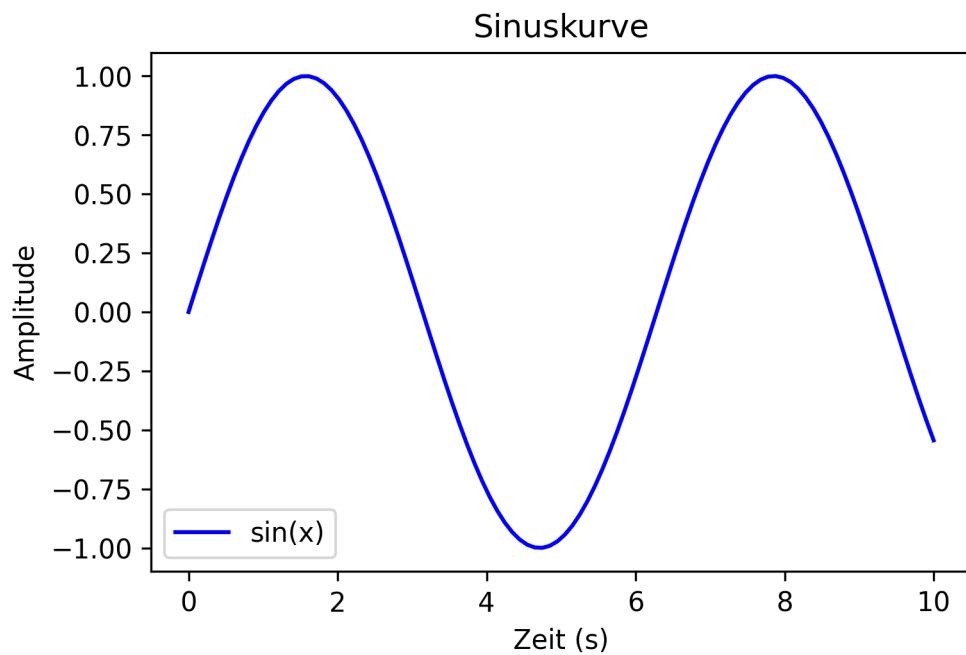
x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y)
plt.show()
```



#### 41.1.2 Besseres Beispiel

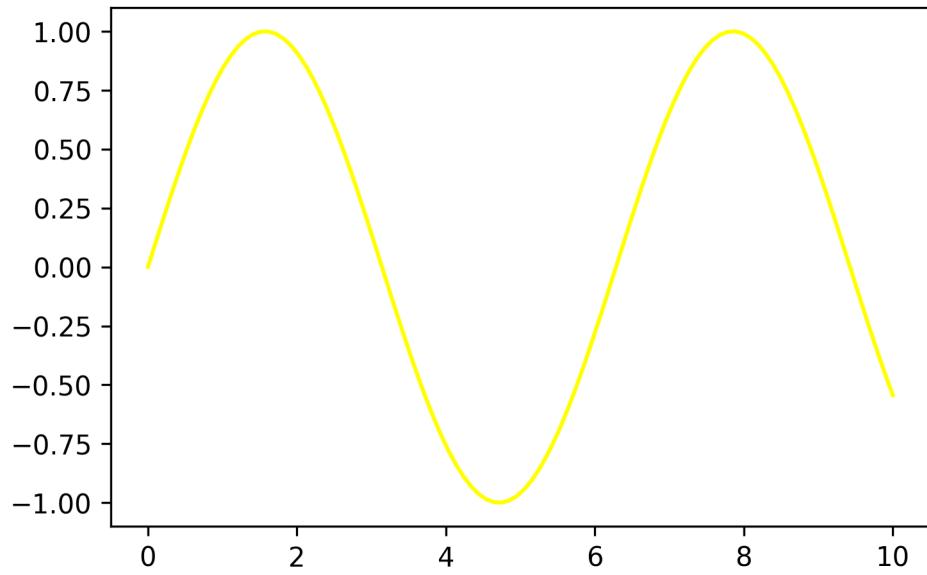
```
plt.plot(x, y, label='sin(x)', color='b')
plt.xlabel('Zeit (s)')
plt.ylabel('Amplitude')
plt.title('Sinuskurve')
plt.legend()
plt.show()
```



## 41.2 2. Ungünstige Farbwahl

### 41.2.1 Schlechtes Beispiel

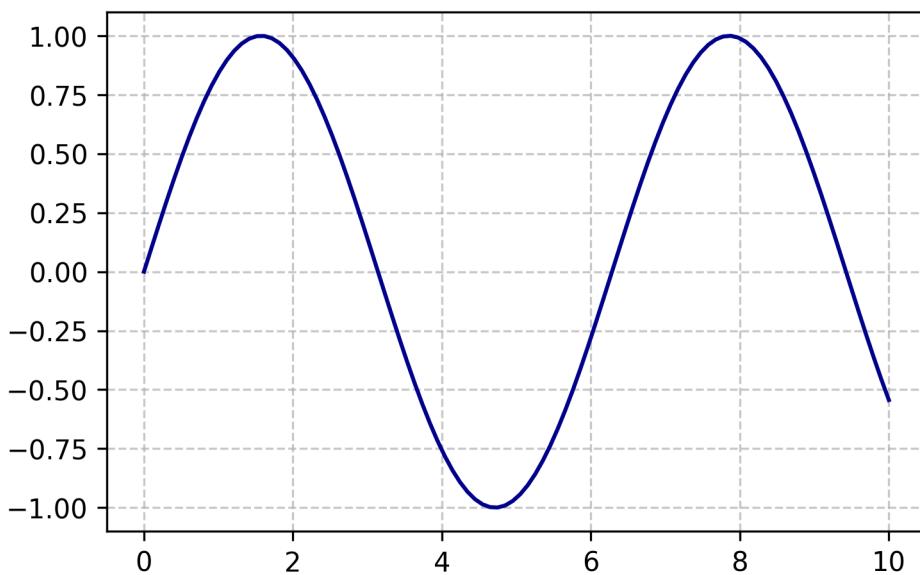
```
plt.plot(x, y, color='yellow')
plt.show()
```



#### 41.2.2 Besseres Beispiel

```
plt.plot(x, y, color='darkblue')
plt.grid(True, linestyle='--', alpha=0.7)
plt.title('Gute Kontraste für bessere Lesbarkeit')
plt.show()
```

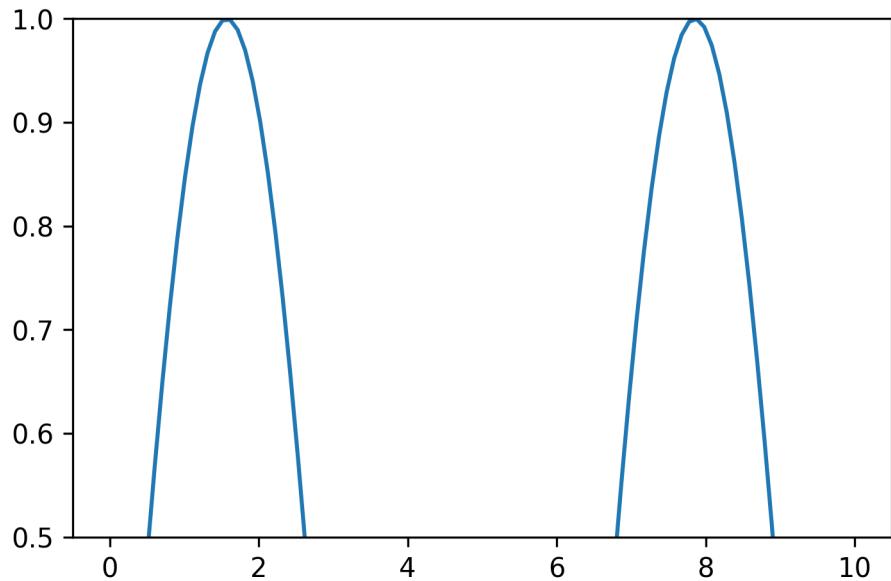
Gute Kontraste für bessere Lesbarkeit



### 41.3 3. Keine sinnvolle Achsen Skalierung

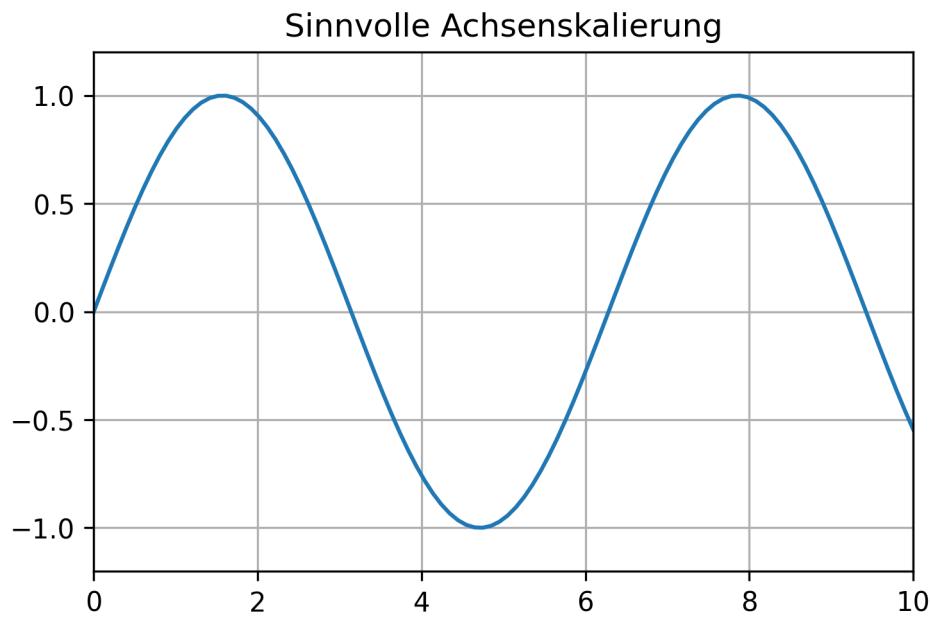
#### 41.3.1 Schlechtes Beispiel

```
plt.plot(x, y)
plt.ylim(0.5, 1)
plt.show()
```



#### 41.3.2 Besseres Beispiel

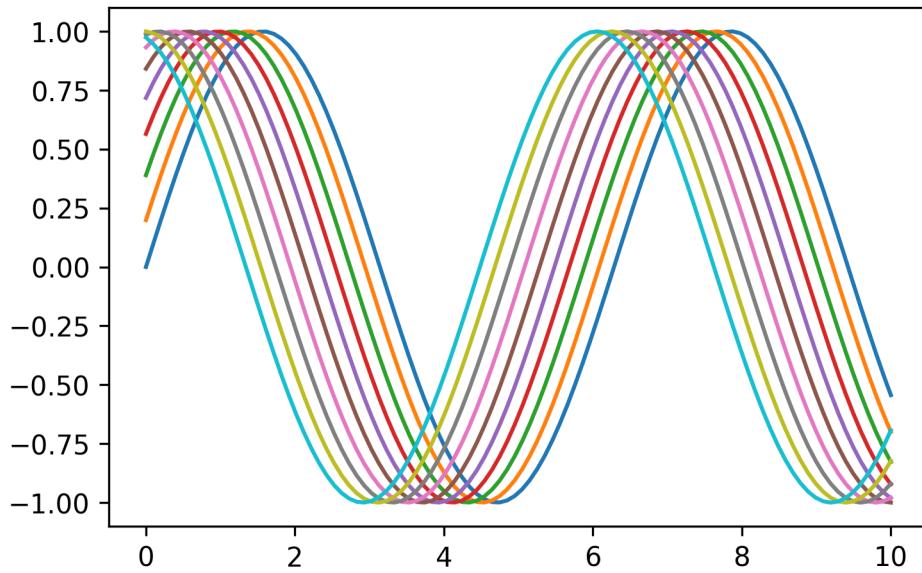
```
plt.plot(x, y)
plt.ylim(-1.2, 1.2)
plt.xlim(0, 10)
plt.grid(True)
plt.title('Sinnvolle Achsenkalierung')
plt.show()
```



## 41.4 4. Überladung durch zu viele Linien

### 41.4.1 Schlechtes Beispiel

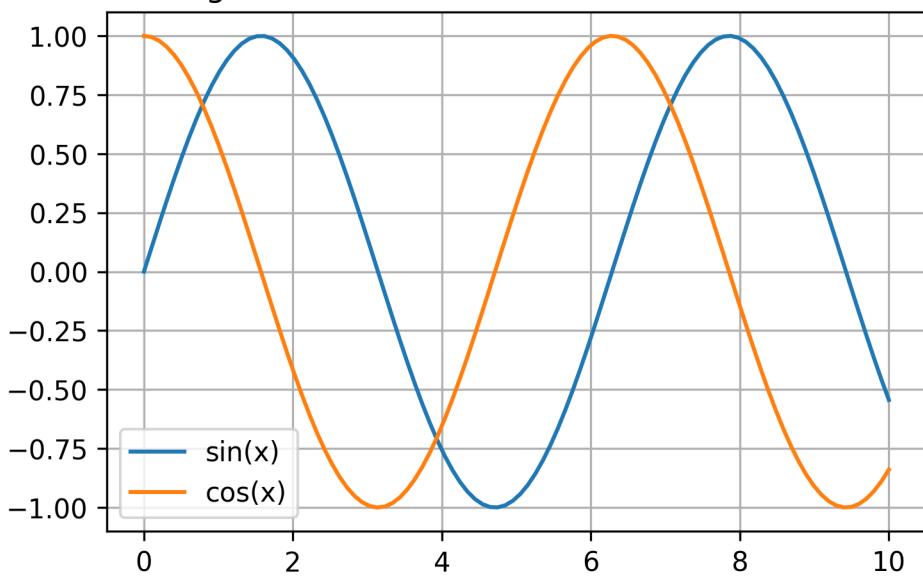
```
for i in range(10):
    plt.plot(x, np.sin(x + i * 0.2))
plt.show()
```



#### 41.4.2 Besseres Beispiel

```
plt.plot(x, np.sin(x), label='sin(x)')
plt.plot(x, np.cos(x), label='cos(x)')
plt.legend()
plt.title('Weniger ist mehr: Reduzierte Informationsdichte')
plt.grid(True)
plt.show()
```

Weniger ist mehr: Reduzierte Informationsdichte



## 41.5 Fazit

# **Part VI**

## **w-python-pandas**

# Werkzeugbaustein Pandas

Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch und Sebastian Seipel. Werkzeugbaustein Pandas von Marc Fehr und Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar auf [GitHub](https://github.com/bausteine-der-datenanalyse/w-pandas). Ausgenommen von der Lizenz sind alle Logos Dritter und anders gekennzeichneten Inhalte. 2025



<https://github.com/bausteine-der-datenanalyse/w-pandas>

## Voraussetzungen

- 
- 
- 
- 
-

- CSV-Datei
- GitHub
- GitHub
- kaggle

## Lernziele

- 
- 
- 
-

## 42 Einleitung

- 
- 
- 

```
import numpy as np  
import pandas as pd
```

### 42.1 Die Datenstrukturen Series und DataFrame

- 
- 

[MultiIndex](#)

#### 42.1.1 Series

```
einzelwert_series = pd.Series('Hallo Welt!')
print(f"Series aus Einzelwert:\n{einzelwert_series}")

numerische_series = pd.Series([1, 2, 3])
print(f"\nSeries aus Liste:\n{numerische_series}")

alphanumerische_series = pd.Series(('a', '5', 'g'))
print(f"\nSeries aus Tupel:\n{alphanumerische_series}")

boolean_series = pd.Series(np.array([True, False, True])) # NumPy-Array
print(f"\nSeries aus NumPy-Array:\n{boolean_series}")
```

- 
- 
- 

```
numerische_series = pd.Series([1, 2, 3], dtype = 'float', index = ['A1', 'B2', 'C3'], name = None)
print(numerische_series)
```

## Dokumentation

Pandas-

```
print(f"Name der Series: {numerische_series.name}")
numerische_series.name = 'Fließkommazahlen'

print(f"Index der Series: {numerische_series.index}")
numerische_series.index = ['eins', 'zwei', 'drei']

numerische_series = numerische_series.astype('string')
print(f"\nDie geänderte Series:\n{numerische_series}")
```

## 42.1.2 Aufgabe Series

💡 Tip ???: Musterlösung dtype

```
numerische_series = pd.Series([1, 2, 3], dtype = 'float', index = ['A1', 'B2', 'C3'], name = 'Ganzzahlen')

# falls numerische_series vorher vom dtype string ist,
# muss erst in dtype float konvertiert werden
# numerische_series = numerische_series.astype('float')

numerische_series = numerische_series.astype('int')

print(numerische_series)

A1    1
B2    2
C3    3
Name: Ganzzahlen, dtype: int64
```

## 42.1.3 DataFrame

```
einzelwert_df = pd.DataFrame(['Hallo Welt!'])
print(einzelwert_df, "\n")

df_aus_listen = pd.DataFrame([[1, 2, 3], [4, 5, 6]])
print(df_aus_listen, "\n")

df_aus_series = pd.DataFrame([alphanumerische_series, boolean_series])
print(df_aus_series, "\n")

df_aus_verschieden = pd.DataFrame([np.array([True, False, True])], alphanumerische_series, [1])
print(df_aus_verschieden)
```

- 
- 
-

```
df_transponiert = pd.DataFrame([[1, 2, 3], [True, False, True]], index = ['Spalte 1', 'Spalte 2'])
print(df_transponiert)
```

```
df_transponiert = pd.DataFrame([[1, 2, 3], [True, False, True]]).T
df_transponiert.columns = ['Spalte 1', 'Spalte 2']
df_transponiert.index = ['Zeile 1', 'Zeile 2', 'Zeile 3']
print(df_transponiert)
```

```
df_transponiert = pd.DataFrame([[1, 2, 3], ['a', 'b', 'c']], index = ['Zahlen', 'Buchstaben'])
print(df_transponiert)
print(f"\n{df_transponiert.dtypes}")
```

```
df = pd.DataFrame({'Spalte 1': [1, 2, 3], 'Spalte 2': [4.1, 5.6, 6.0]}, index = ['oben', 'mitte', 'unten'])
print(df)

# einen leeren DataFrame erzeugen
df = pd.DataFrame()

# Zuweisung von Daten
df['Spaltenbeschriftung'] = [1, 2, 3]
df['zweite Spalte'] = alphanumerische_series

print(df)
```

#### 💡 Tip ??: Der Index

In den meisten Fällen ist der von 0 bis n-1 reichende Index am praktischsten. Der numerische Index hilft bei der Auswahl von Indexbereichen (Slicing) und der Arbeit mit mehreren Datenstrukturen. Probieren Sie einmal aus, was passiert, wenn Sie einen DataFrame aus zwei Series mit unterschiedlichen Indizes erstellen.  
Auch widerspricht das Auslagern beschreibender oder gemessener Variablen in den Index dem Konzept tidy data, einem System zum Strukturieren von Datensätzen, das Sie im Methodenbaustein [Einlesen strukturierter Datensätze](#) kennenlernen.

```
df = pd.DataFrame({'Spalte 1': ['1', '2', '3'], 'Spalte 2': [True, False, True]})  
print(f"Die Datentypen von df:\n{df.dtypes}")
```

```
# Datentyp von Spalte 1 ändern  
df['Spalte 1'] = df['Spalte 1'].astype('string')  
print(f"\nDie Datentypen von df:\n{df.dtypes}")
```

```
df = df.astype('string')  
print(f"\nDie Datentypen von df:\n{df.dtypes}")
```

```
df = df.astype({'Spalte 1': 'int', 'Spalte 2': 'bool'})  
print(f"\nDie Datentypen von df:\n{df.dtypes}")
```

```
# ändern der Spaltennamen über das Attribut .columns  
df.columns = ['Spalte1', 'Spalte2']  
df.index = [1, 2, 3]  
print(df)
```

```
df.rename(columns = {'Spalte1': 'Spalte_1', 'Spalte2': 'Spalte_2'}, index = {1: 'A1', 2: 'B2'}  
print(df)
```

```
df.reset_index(inplace = True, drop = True)  
print(df)
```

#### 42.1.4 Aufgabe DataFrame

- 
- 

💡 Tip ???: Musterlösung

```
ferien = [False, False, False, True, False, True, True, True, False, True, False, True]

df = pd.DataFrame({
    'Nummer': list(range(1,13)),
    'Monat': ['Januar', 'Februar', 'März', 'April', 'Mai', 'Juni', 'Juli', 'August', 'September',
              'Oktober', 'November', 'Dezember'],
})

df['Ferien'] = ferien

print(df)
```

	Nummer	Monat	Ferien
0	1	Januar	False
1	2	Februar	False
2	3	März	False
3	4	April	True
4	5	Mai	False
5	6	Juni	True
6	7	Juli	True
7	8	August	True
8	9	September	False
9	10	Oktober	True
10	11	November	False
11	12	Dezember	True

## 42.2 Deskriptive Datenanalyse mit Pandas

**Listing 42.1**

```
dateipfad = "01-daten/ToothGrowth.csv"
meerschweinchen = pd.read_csv(filepath_or_buffer = dateipfad, sep = ',', header = 0, \
    names = ['ID', 'len', 'supp', 'dose'], dtype = {'ID': 'int', 'len': 'float', 'dose': 'float'})
```

<https://doi.org/10.1093/jn/33.5.491>

```
meerschweinchen.info()
```

```
print(meerschweinchen.shape)
```

```
print(meerschweinchen.describe(include = 'all'))
```

```
print(meerschweinchen.describe(include = ['float']))
```

```
print(meerschweinchen.describe(include = ['category']))
```

```
meerschweinchen.count(axis = 'rows') # der Standardwert von axis ist 'rows'
```

```
meerschweinchen['dose'].value_counts()
```

```
meerschweinchen['dose'].unique()
```

## 42.3 Slicing

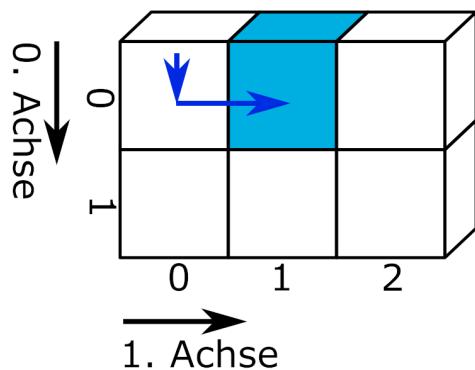


Figure 42.1: zweidimensionaler Datensatz

[CC-BY-4.0](#)

[GitHub](#)

#### 42.3.1 Slice Operator

```
zehn_zahlen = pd.Series(range(0, 10))
print(zehn_zahlen[3:6])
```

```
print(meerschweinchen[7:12])
```

```
print(meerschweinchen['dose'][10:15], "\n")
print(type(meerschweinchen['dose'][10:15]))
```

### **⚠ Warning ??: Verkettete Indexierung**

Die verkettete Indexierung erzeugt in Pandas abhängig vom Kontext eine Kopie des Objekts oder greift auf den Speicherbereich des Objekts zu. Mit Pandas 3.0 wird die verkettete Indexierung nicht mehr unterstützt, das Anlegen einer Kopie wird zum Standard werden. Weitere Informationen erhalten Sie im zitierten Link.

“Whether a copy or a reference is returned for a setting operation, may depend on the context. This is sometimes called `chained assignment` and should be avoided. See [Returning a View versus Copy](#).”

([Pandas Dokumentation](#))

#### **42.3.2 Slicing mit Pandas-Methoden**

- 
- 

##### **42.3.2.1 Beschriftungsbasiertes Slicing mit .loc[]**

### **⚠ Warning ??: inklusives Slicing**

Anders als die Pythonbasis und das Slicing mit `.iloc[]` zählt Pandas beim beschriftungsbasiertem Slicing inklusiv, gibt also die letzte ausgewählte Position mit aus.

```

# Nummern
zehn_zahlen = pd.Series(range(0, 10))
print("Rückgabe eines Einzelwerts:", zehn_zahlen.loc[5]) # Einzelwert
print(zehn_zahlen.loc[[2, 4, 7]]) # Liste
print(zehn_zahlen.loc[5:7], "\n") # Slice

# Buchstaben und andere Zeichen
sechs_zahlen = pd.Series(list(range(0, 6)), index = ['a', 'b', 'c', 'd', 'e', 'f'])
print("Rückgabe eines Einzelwerts:", sechs_zahlen.loc['c']) # Einzelwert
print(sechs_zahlen.loc[['c', 'f', 'a']]) # Liste
print(sechs_zahlen.loc['c':'e']) # Slice

```

```

try:
    print(sechs_zahlen.loc[2:4])
except Exception as error:
    print(error)

print("\n", sechs_zahlen.loc['c':'e'], sep = "\n")

```

```
print(meerschweinchen.loc[18:22, ['len', 'dose']])
```

#### 42.3.3 Indexbasiertes Slicing mit .iloc[]

##### ⚠ Warning ???: exklusives Slicing

Beim Slicing mit der Methode .iloc[] zählt Pandas wie die Pythonbasis exklusiv.

```
print("Rückgabe eines Einzelwerts:", meerschweinchen.iloc[27, 2]) # Einzelwerte
print(meerschweinchen.iloc[[27, 29, 52], 2:4]) # Liste und Slice
```

#### 42.3.3.1 Die Methoden `.head()` und `.tail()`

```
print(meerschweinchen.head(3), "\n")
print(meerschweinchen.tail(3))
```

```
print(meerschweinchen['len'].tail(3))
```

## 42.4 Aufgaben Slicing

```
temperaturen_2021 = pd.Series([2, 4, 7, 12, 19, 23, 25, 23, 18, 15, 9, 5],
                               index = ['Jan', 'Feb', 'Mär', 'Apr', 'Mai', 'Jun',
                               'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez'])
```

1.

2.

3.

4.

### 💡 Tip ???: Musterlösung Slicing

Aufgabe 1

```
print(temperaturen_2021.loc[['Mär', 'Apr', 'Mai']])
```

```
Mär      7
Apr     12
Mai     19
dtype: int64
```

Aufgabe 2

```
print(temperaturen_2021[-3:], "\n")
print(temperaturen_2021.iloc[-3:])
```

```
Okt     15
Nov      9
Dez      5
dtype: int64
```

```
Okt     15
Nov      9
Dez      5
dtype: int64
```

Aufgabe 3

```
print(meerschweinchen.loc[ :, ['dose', 'len']].head(n = 4), "\n")
print(meerschweinchen.loc[ :, ['dose', 'len']].tail(n = 3))
```

```
dose    len
```

```
0    0.5    4.2
1    0.5   11.5
2    0.5    7.3
3    0.5    5.8
```

```
      dose   len
57    2.0  27.3
58    2.0  29.4
59    2.0  23.0
```

#### Aufgabe 4

```
# Slice aus Series
# print(meerschweinchen['dose'].loc[meerschweinchen['dose'] == 2.0])

# Slice aus DataFrame
print(meerschweinchen.loc[meerschweinchen['dose'] == 2.0, ['dose']])
```

```
dose
20    2.0
21    2.0
22    2.0
23    2.0
24    2.0
25    2.0
26    2.0
27    2.0
28    2.0
29    2.0
50    2.0
51    2.0
52    2.0
53    2.0
54    2.0
55    2.0
56    2.0
57    2.0
58    2.0
59    2.0
```

## 42.5 Datenstrukturen verbinden

- 
- 

```
series_1 = pd.Series([1, 2])
series_2 = pd.Series([4, 5])
print(pd.concat([series_1, series_2]), "\n")
print(pd.concat([series_1, series_2], ignore_index = True), "\n")
print(pd.concat([series_1, series_2], ignore_index = True, axis = 1))
```

```
temperaturen_2021 = pd.Series([2, 4, 7, 12, 19, 23, 25, 23, 18, 15, 9, 5],
                               index = ['Jan', 'Feb', 'Mär', 'Apr', 'Mai', 'Jun',
                               'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez'])

temperaturen_2022 = pd.Series([3, 6, 9, 13, 18, 21, 24, 23, 19, 14, 8, 4],
                               index = ['Jan', 'Feb', 'Mär', 'Apr', 'Mai', 'Jun',
                               'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez'])
```

```

temperaturen_2023 = pd.Series([-3, -1, 4, 9, 15, 20, 20, 19, 16, 15, 7, 6],
                             index = ['Jan', 'Feb', 'Mär', 'Apr', 'Mai', 'Jun',
                             'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez'])

temperaturen_2024 = pd.Series([-1, 2, 5, 8, 17, 24, 25, 20, 17, 14, 9, 2],
                             index = ['Jan', 'Feb', 'Mär', 'Apr', 'Mai', 'Jun',
                             'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez'])

# Series zu DataFrame verbinden
df1 = pd.concat([temperaturen_2021, temperaturen_2022], axis = 1)
df2 = pd.concat([temperaturen_2023, temperaturen_2024], axis = 1)

# DataFrames verbinden
temperaturen = pd.concat([df1, df2], axis = 1)
temperaturen.columns = [2021, 2022, 2023, 2024]
print(temperaturen)

```

## 42.6 Einfügen und Löschen in Datenstrukturen

•

•

•

## 42.7 Aufgaben verbinden und löschen

1.

2.

3.

💡 Tip ??: Musterlösung verbinden und löschen

### 1. Aufgabe

```
df = pd.DataFrame()

# Alternative 1
df['len'] = meerschweinchen['len']

# Alternative 2
df.insert(loc = 1, column = 'dose', value = meerschweinchen['dose'])

print(df.head(), "\n", df.shape)

len  dose
```

```
0    4.2    0.5
1   11.5    0.5
2    7.3    0.5
3    5.8    0.5
4    6.4    0.5
(60, 2)
```

### 2. Aufgabe

```
df = df.drop(index = range(1, len(df), 2))

print(df.head(), "\n", df.shape)
```

```
      len  dose
0    4.2  0.5
2    7.3  0.5
4    6.4  0.5
6   11.2  0.5
8    5.2  0.5
(30, 2)
```

### 3. Aufgabe

```
df.insert(loc = 0, column = 'ID', value = meerschweinchen.loc[df.index, 'ID'])

print(df.head(), "\n")
print(df.tail(), "\n")
print("df.shape:", df.shape)
```

```
      ID  len  dose
0     1  4.2  0.5
2     3  7.3  0.5
4     5  6.4  0.5
6     7 11.2  0.5
8     9  5.2  0.5
```

```
      ID  len  dose
50   51 25.5  2.0
52   53 22.4  2.0
54   55 24.8  2.0
56   57 26.4  2.0
```

```
58 59 29.4 2.0
```

```
df.shape: (30, 3)
```

## Quellen

[https://pandas.pydata.org/docs/user\\_guide/dsintro.html](https://pandas.pydata.org/docs/user_guide/dsintro.html)   [https://pandas.pydata.org/docs/user\\_guide/basics.html](https://pandas.pydata.org/docs/user_guide/basics.html)

## 43 Operationen

```
print("Temperaturen in Celsius:")
print(27 * "=")
print(temperaturen, "\n")

print("Temperaturen in Fahrenheit:")
print(27 * "=")
print(temperaturen * 9/5 + 32)
```

```
print("Minusgrade:")
print(27 * "=")
print(temperaturen < 0)
```

### 43.1 Zeilen- und spaltenweise Operationen

<https://pandas.pydata.org/docs/reference/index.html>

### 43.1.1 arithmetische Funktionen

```
print("Temperaturen in Fahrenheit:")
print(27 * "=")
print(temperaturen.mul(9).div(5).add(32))
```

Methodenbaustein Einlesen strukturierter  
Datensätze

```
missing_value = pd.Series([1, pd.NA, 3])
print(missing_value.add(1, fill_value = -999), "\n")
print(missing_value.add(1, fill_value = np.nan), "\n")
print(missing_value.add(1, fill_value = pd.NA))
```

### 43.1.2 summarische Funktionen

- 
- 
- 
- 
- 
- 
- 

```
# spaltenweise
print("Mittlere Jahrestemperaturen")
print(27 * "=")
print(temperaturen.mean(), "\n")

# zeilenweise
print("Monatliche Mindesttemperatur")
print(28 * "=")
print(temperaturen.min(axis = 1))
```

### 43.1.3 boolsche Funktionen

- 
- 

**i** Note ???: klassenabhängige Funktionsausführung

## 43.2 Einzelwerte oder Liste

Für Einzelwerte oder eine Liste wird die Übereinstimmung elementweise überprüft.

```
print(temperaturen, "\n")  
print(temperaturen.isin([2, 3]))
```

	2021	2022	2023	2024
Jan	2	3	-3	-1
Feb	4	6	-1	2
Mär	7	9	4	5
Apr	12	13	9	8
Mai	19	18	15	17
Jun	23	21	20	24
Jul	25	24	20	25
Aug	23	23	19	20
Sep	18	19	16	17
Okt	15	14	15	14
Nov	9	8	7	9
Dez	5	4	6	2

	2021	2022	2023	2024
Jan	True	True	False	False
Feb	False	False	False	True
Mär	False	False	False	False
Apr	False	False	False	False
Mai	False	False	False	False
Jun	False	False	False	False
Jul	False	False	False	False
Aug	False	False	False	False
Sep	False	False	False	False
Okt	False	False	False	False
Nov	False	False	False	False
Dez	False	False	False	True

### 43.3 NumPy-Array

Für ein NumPy-Array wird die Übereinstimmung elementweise überprüft (vergleiche zum nächsten Reiter).

```
print(type(temperaturen[2021].values), "\n")
print(temperaturen.isin(temperaturen[2021].values))
```

```
<class 'numpy.ndarray'>
2021    2022    2023    2024
Jan  True  False  False  False
```

Feb	True	False	False	True
Mär	True	True	True	True
Apr	True	False	True	False
Mai	True	True	True	False
Jun	True	False	False	False
Jul	True	False	False	True
Aug	True	True	True	False
Sep	True	True	False	False
Okt	True	False	True	False
Nov	True	False	True	True
Dez	True	True	False	True

## 43.4 Series

Für eine Series wird die Übereinstimmung positionsweise geprüft (vergleiche zum vorherigen Reiter). Der Index muss übereinstimmen.

```
print(temperaturen.isin(temperaturen[2021]), "\n")

temperaturen_2021_falscher_index = pd.Series([2, 4, 7, 12, 19, 23, 25, 23, 18, 15, 9, 5])
temperaturen_2021_falscher_index.index = ['A', 'B', 'C', 'D', 'E', 'F', 'Jul', 'Aug', 'Sep']

print("Der Index der Series lautet:\n['A', 'B', 'C', 'D', 'E', 'F', 'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez']")
print(temperaturen.isin(temperaturen_2021_falscher_index))
```

	2021	2022	2023	2024
Jan	True	False	False	False
Feb	True	False	False	False
Mär	True	False	False	False
Apr	True	False	False	False
Mai	True	False	False	False
Jun	True	False	False	False
Jul	True	False	False	True
Aug	True	True	False	False
Sep	True	False	False	False
Okt	True	False	True	False
Nov	True	False	False	True
Dez	True	False	False	False

Der Index der Series lautet:

['A', 'B', 'C', 'D', 'E', 'F', 'Jul', 'Aug', 'Sep', 'Okt', 'Nov', 'Dez'].

Das Ergebnis an den Indexpositionen A-F ist immer False.

	2021	2022	2023	2024
Jan	False	False	False	False
Feb	False	False	False	False
Mär	False	False	False	False
Apr	False	False	False	False
Mai	False	False	False	False
Jun	False	False	False	False
Jul	True	False	False	True
Aug	True	True	False	False
Sep	True	False	False	False
Okt	True	False	True	False
Nov	True	False	False	True
Dez	True	False	False	False

## 43.5 DataFrame

Für einen DataFrame wird die Übereinstimmung positionsweise geprüft. Index und Spaltennamen müssen übereinstimmen (Index siehe Reiter Series).

```
temperaturen_2021_df = pd.DataFrame(temperaturen[2021])
print(temperaturen.isin(temperaturen_2021_df), "\n")

temperaturen_2021_df.columns = [2035]
print(temperaturen.isin(temperaturen_2021_df), "\n")
```

	2021	2022	2023	2024
Jan	True	False	False	False
Feb	True	False	False	False
Mär	True	False	False	False
Apr	True	False	False	False
Mai	True	False	False	False
Jun	True	False	False	False
Jul	True	False	False	False
Aug	True	False	False	False
Sep	True	False	False	False
Okt	True	False	False	False
Nov	True	False	False	False
Dez	True	False	False	False

2021 2022 2023 2024

Jan	False	False	False	False
Feb	False	False	False	False
Mär	False	False	False	False
Apr	False	False	False	False
Mai	False	False	False	False
Jun	False	False	False	False
Jul	False	False	False	False
Aug	False	False	False	False
Sep	False	False	False	False
Okt	False	False	False	False
Nov	False	False	False	False
Dez	False	False	False	False

💡 Tip ???: Überraschungen vermeiden

Eine klassenabhängige Funktionsausführung kann, wenn das Verhalten unbemerkt bleibt, die Ergebnisse einer Datenanalyse verfälschen. Um dies zu verhindern, sollten Sie 3 allgemeine Ratschläge befolgen:

1. Schauen Sie in die Dokumentation der jeweiligen Funktion. Python und viele Module entwickeln sich dynamisch, sodass sich das Verhalten einer Funktion verändern kann.
2. Gehen Sie schrittweise vor und lassen sich die Zwischenergebnisse von Arbeitsschritten mit der Funktion `print()` ausgeben.
3. Bei großen Datenmengen ist es häufig einfacher, mit eigens erzeugten Testdaten zu arbeiten. Ein zehnzeiliger DataFrame mit den Datentypen und der Struktur der Arbeitsdaten, ist leichter zu überblicken. Nutzen Sie einen solchen Testdatensatz um die von Ihnen verwendeten Funktionen zu überprüfen.

```
print(temperaturen.le(2), "\n")
print(temperaturen[2021].gt(5))
```

#### 43.5.1 Verwendung der Methoden .agg() und .apply()

## 43.6 Funktion

```
def my_plus_ten(x):
    y = x + 10
    return y

print(temperaturen.agg(my_plus_ten), "\n")
print(temperaturen.apply(my_plus_ten))
```

## 43.7 Funktionsname

```
print(temperaturen.agg("sum"), "\n")
print(temperaturen.apply("sum"))
```

## 43.8 Liste von Funktionen

```
print(temperaturen.agg(["sum", "mean", "median"]), "\n")
print(temperaturen.apply(["sum", "mean", "median"]))
```

## 43.9 Dictionary von Funktionen

```
print(temperaturen.agg({2021: "sum", 2022: "mean", 2023: "median", 2024: "min"}), "\n")
print(temperaturen.apply({2021: "sum", 2022: "mean", 2023: "median", 2024: "min"}), "\n")
```

```
# Auf die Series angewendet
print(len(str(temperaturen[2021])), "\n")

# Elementweise angewendet
print(temperaturen[2021].agg(lambda x: len(str(x))), "\n") # deprecated
print(temperaturen[2021].apply(lambda x: len(str(x))), "\n")
```

## Lambda-Ausdrucks

in  
diesem Artikel

```
# print(temperaturen[2021].map(lambda x: len(str(x))))
# print(temperaturen[2021].transform(lambda x: len(str(x))), "\n")
```

## 43.10 Aufgaben Operationen

```
# Temperaturdaten
temperaturen_2021 = pd.Series([2, 4, 7, 12, 19, 23, 25, 23, 18, 15, 9, 5])
temperaturen_2022 = pd.Series([3, 6, 9, 13, 18, 21, 24, 23, 19, 14, 8, 4])
temperaturen_2023 = pd.Series([-3, -1, 4, 9, 15, 20, 20, 19, 16, 15, 7, 6])
temperaturen_2024 = pd.Series([-1, 2, 5, 8, 17, 24, 25, 20, 17, 14, 9, 2])

# DataFrame erzeugen
temperaturen = pd.concat([temperaturen_2021, temperaturen_2022, temperaturen_2023, temperaturen_2024])
temperaturen.columns = [2021, 2022, 2023, 2024]
temperaturen.index = ['Jan', 'Feb', 'Mär', 'Apr', 'Mai', 'Jun', 'Jul', 'Aug', 'Sep', 'Okt',
```

- 1.
- 2.
- 3.

💡 Tip ??: Musterlösung Aufgaben Operationen

### 1. Aufgabe

```
print(temperaturen.mean(axis = 1))
```

```
Jan      0.25
Feb     2.75
Mär     6.25
Apr    10.50
Mai    17.25
Jun    22.00
Jul    23.50
Aug    21.25
Sep    17.50
Okt    14.50
Nov     8.25
Dez     4.25
dtype: float64
```

### 2. Aufgabe

```
print(temperaturen.mean(axis = 1).ge(21))
```

```
Jan      False
Feb      False
Mär      False
Apr      False
Mai      False
Jun      True
Jul      True
Aug      True
Sep      False
Okt      False
Nov      False
Dez      False
dtype: bool
```

### 3. Aufgabe

```
print(temperaturen.index[temperaturen.mean(axis = 1).ge(21)], "\n")  
  
# als Liste  
print(list(temperaturen.index[temperaturen.mean(axis = 1).ge(21)]), "\n")  
  
Index(['Jun', 'Jul', 'Aug'], dtype='object')  
['Jun', 'Jul', 'Aug']
```

## 43.11 Suchen und ersetzen

```
print(np.where(temperaturen == 4))
```

```
print(temperaturen.iloc[1, 0])  
print(temperaturen.iloc[2, 2])
```

- 
-

```
print(temperaturen.replace(to_replace = 25, value = 1000), "\n")
print(temperaturen.where(temperaturen == 25, other = 1000))
```

### 43.12 Aufgaben suchen und ersetzen

1.

2.

💡 Tip ??: Musterlösung suchen und ersetzen

### 1. Aufgabe

```
print(np.where(temperaturen <= 0))
print("Anzahl Werte:", len(np.where(temperaturen <= 0)[0]))

for i in range(len(np.where(temperaturen <= 0)[0])):
    print(temperaturen.iloc[np.where(temperaturen <= 0)[0][i], np.where(temperaturen <= 0)[1][i]])

(array([0, 0, 1]), array([2, 3, 2]))
Anzahl Werte: 3
-3
-1
-1
```

### 2. Aufgabe

```
print(temperaturen.where(temperaturen > 0, other = 0))
```

	2021	2022	2023	2024
Jan	2	3	0	0
Feb	4	6	0	2
Mär	7	9	4	5
Apr	12	13	9	8
Mai	19	18	15	17
Jun	23	21	20	24
Jul	25	24	20	25
Aug	23	23	19	20
Sep	18	19	16	17
Okt	15	14	15	14
Nov	9	8	7	9
Dez	5	4	6	2

## 43.13 Sortieren

```
print(temperaturen.sort_index(), "\n")
print(temperaturen.sort_index(axis = 1, ascending = False))
```

[laut Dokumentation](#)

```
# Sortieren nach numerischen Spaltenbeschriftungen
print(temperaturen.sort_values(by = 2021), "\n")
print(temperaturen.sort_values(by = [2021, 2023]), "\n")

# Sortieren nach als string übergebenen Spaltenbeschriftungen
# führt zu KeyError, die Fehlermeldung wird nicht vollständig abgefangen
try:
    print(temperaturen.sort_values(by = '2021'))
except Exception as error:
    print(error)
```

## 43.14 Aufgaben Sortieren

1.

2.

💡 Tip ??: Musterlösung Sortieren

1. Aufgabe

```
print(meerschweinchen.sort_values(by = 'len', ascending = False).head(), "\n")  
  
print("Die ID lautet:", meerschweinchen.sort_values(by = 'len', ascending = False).iloc[0,  
  
ID    len supp  dose  
22   23  33.9   VC   2.0  
25   26  32.5   VC   2.0  
55   56  30.9   OJ   2.0  
29   30  29.5   VC   2.0  
58   59  29.4   OJ   2.0
```

Die ID lautet: 23

2. Aufgabe

```
dose_1 = meerschweinchen[meerschweinchen['dose'] == 1.0]  
  
print(dose_1.sort_values(by = 'len', ascending = False).head(), "\n")  
  
print("Die ID lautet:", dose_1.sort_values(by = 'len', ascending = False).iloc[0, 0])  
  
ID    len supp  dose  
49   50  27.3   OJ   1.0  
43   44  26.4   OJ   1.0  
46   47  25.8   OJ   1.0  
45   46  25.2   OJ   1.0  
42   43  23.6   OJ   1.0
```

Die ID lautet: 50

## 43.15 GroupBy

1.

2.

3.

4.

## 43.16 DataFrame meerschweinchen

```
print(meerschweinchen.head(n = 12))
```

### **43.17 meerschweinchen gruppiert nach Verabreichungsart**

```
print(meerschweinchen.groupby('supp').head(n = 6))
```

### **43.18 Länge nach Verabreichungsart**

```
print(meerschweinchen.groupby(by = 'supp')['len'].mean())
```

### **43.19 Länge nach Verabreichungsart und Dosis**

```
print(meerschweinchen.groupby(by = ['supp', 'dose'])['len'].mean())
```

## 43.20 Aufgaben GroupBy

**Listing 43.1**

```
mtcars = pd.read_csv(filepath_or_buffer = "01-daten/mtcars.csv", sep = ",")  
mtcars.rename(columns = {'Unnamed: 0': 'car'}, inplace = True)  
  
mtcars.head()
```

	car	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
1	Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
2	Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
3	Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
4	Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

1.

2.

3.

💡 Tip ???: Musterlösung GroupBy

1. Aufgabe

```
mtcars.groupby(by = 'cyl')['mpg'].mean()
```

```
cyl
4    26.663636
6    19.742857
8    15.100000
Name: mpg, dtype: float64
```

2. Aufgabe

```
# 1 Meile = 1.60934 Kilometer
# 1 Gallone = 3.78541 Liter

mpg = mtcars.groupby(by = 'cyl')['mpg'].mean()

liter_100km = 1 / mpg.mul(1.60934).div(3.78541).div(100)

print(liter_100km)
```

```
cyl
4     8.821567
6    11.913932
8    15.577156
Name: mpg, dtype: float64
```

3. Aufgabe

```
print(mtcars.groupby(by = ['cyl', 'carb'])['qsec'].mean(), "\n")
print(mtcars.groupby(by = ['cyl', 'carb'])['qsec'].mean().index[-1], "\n")
```

```
cyl  carb
4    1      19.378000
     2      18.936667
6    1      19.830000
     4      17.670000
     6      15.500000
8    2      17.060000
     3      17.666667
     4      16.495000
     8      14.600000
Name: qsec, dtype: float64

(np.int64(8), np.int64(8))
```

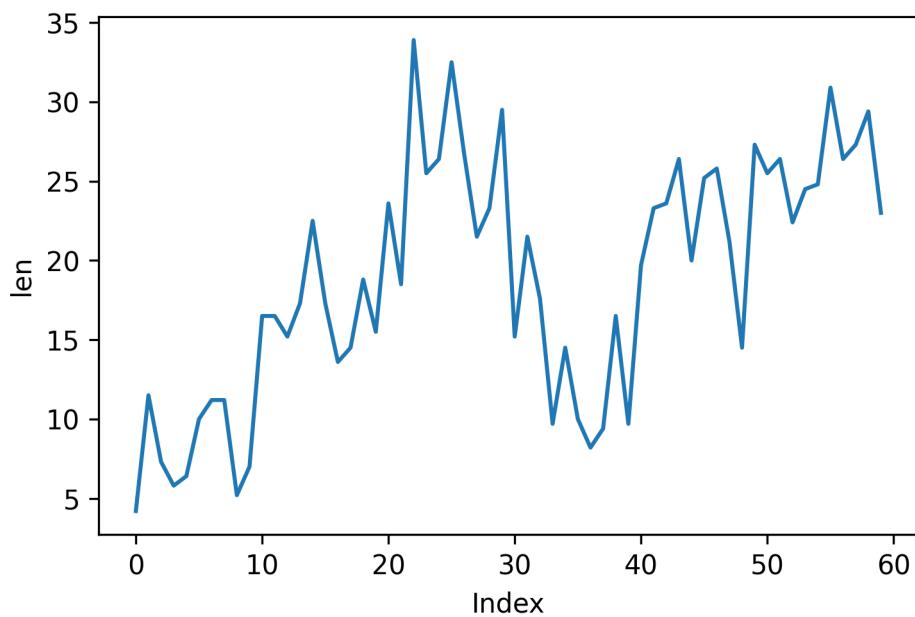
Die Gruppe mit 8 Zylindern und 8 Vergasern ist am schnellsten. (Hinweis: Es handelt sich hierbei um einen sogenannten [MultiIndex](#).)

# 44 Grafikerstellung

Werkzeugbaustein Matplotlib

## 44.1 Series

```
meerschweinchen['len'].plot(xlabel = 'Index', ylabel = 'len')
```



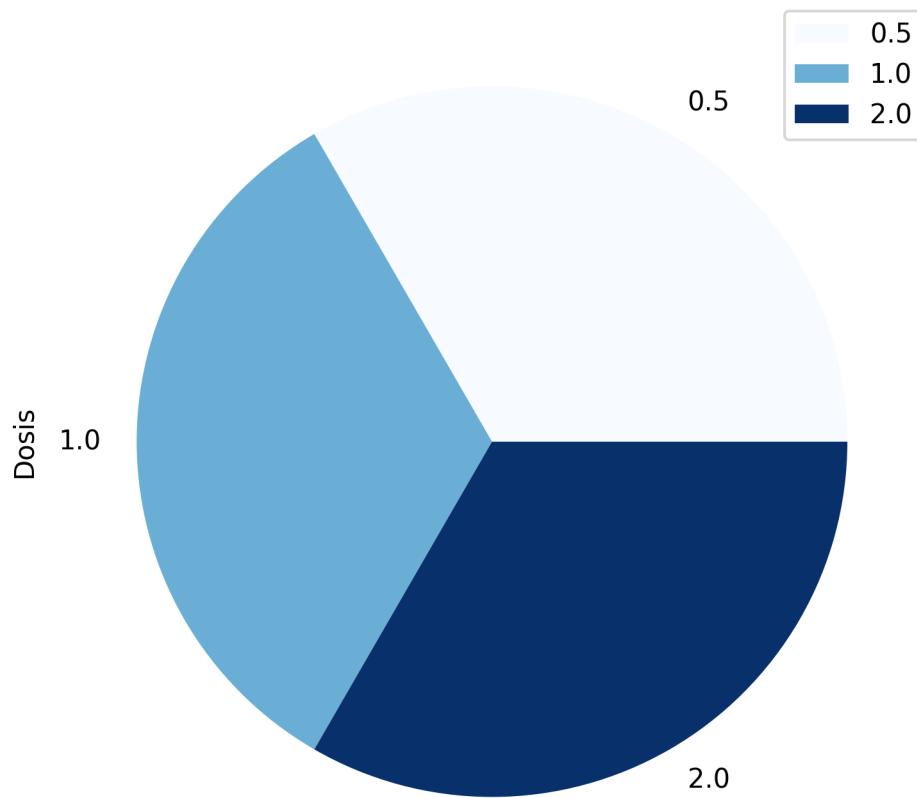
•  
•  
•  
•  
•

•  
•  
•  
•  
•

## Dokumentation

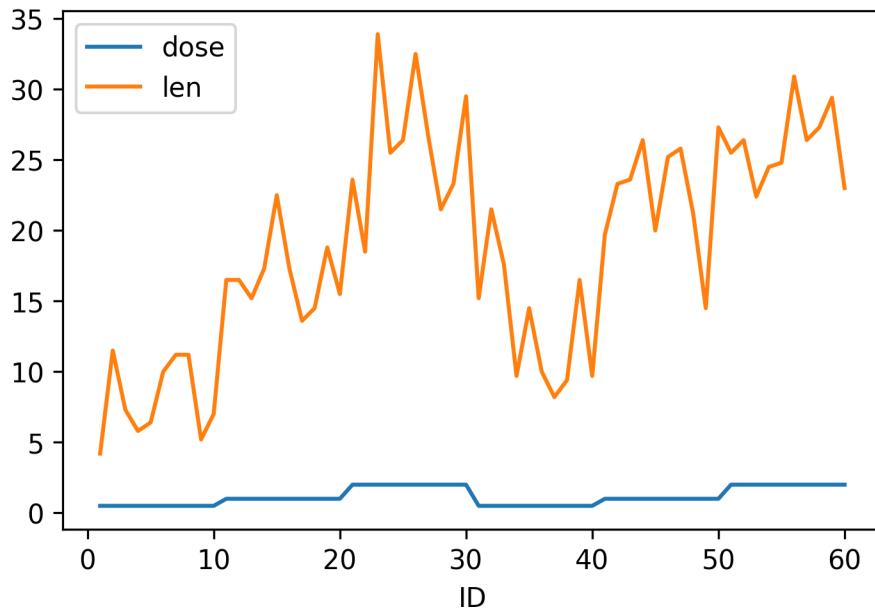
```
meerschweinchen['dose'].value_counts().plot(kind = 'pie', ylabel = 'Dosis', colormap = 'Blues')
```

Tortendiagramm der Dosis Vitamin C

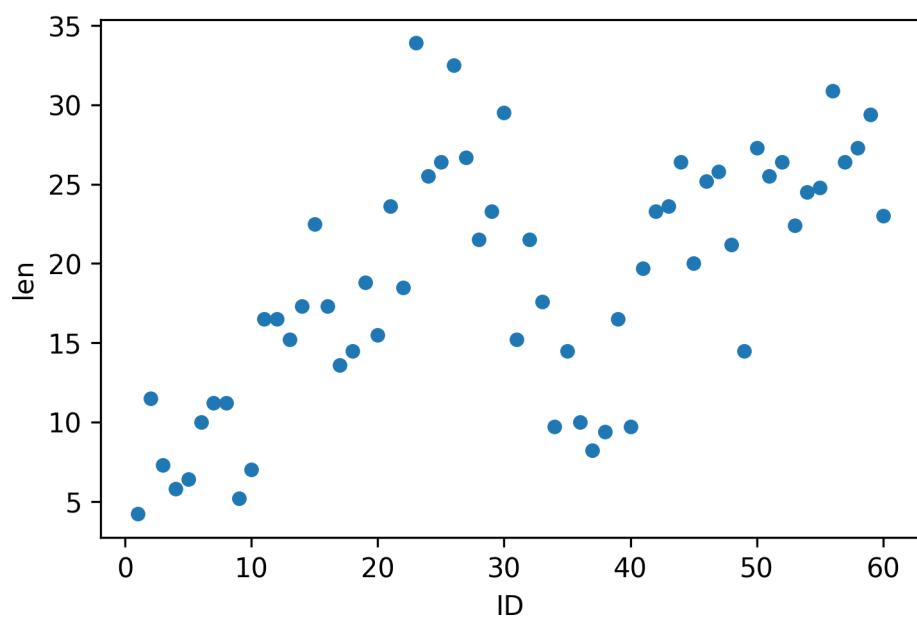


## 44.2 DataFrame

```
meerschweinchen.plot(x = 'ID', y = ['dose', 'len'])
```

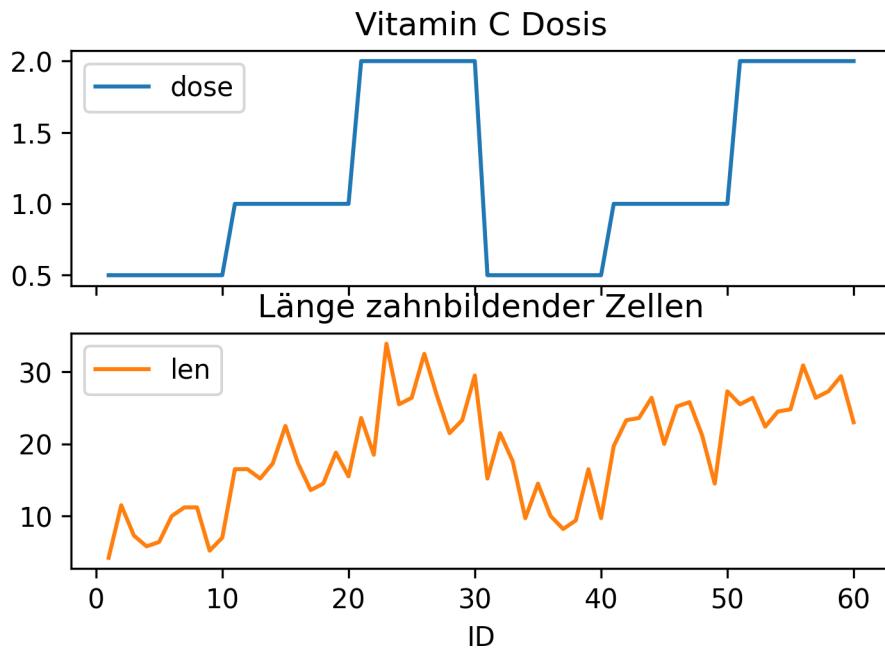


```
meerschweinchen.plot(x = 'ID', y = 'len', kind = 'scatter')
```



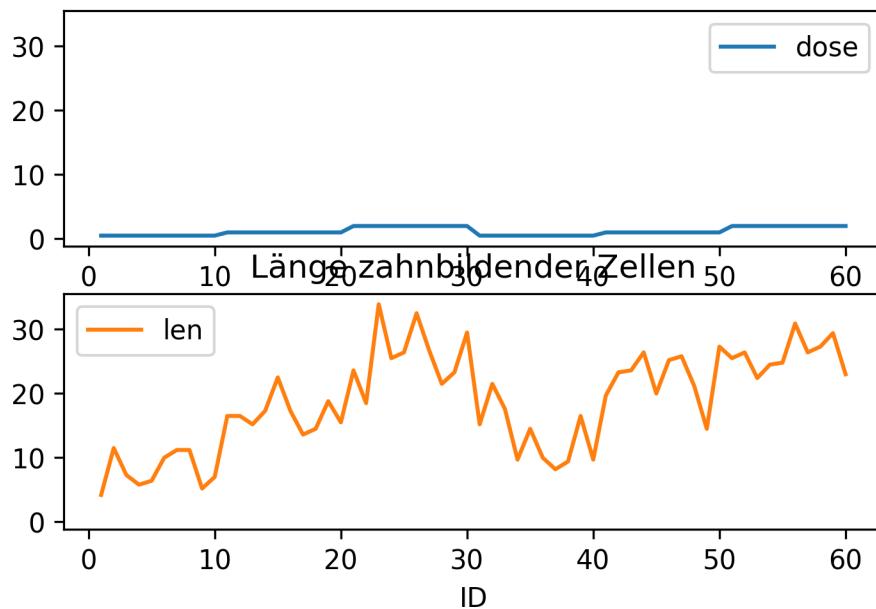
## 44.3 subplots

```
meerschweinchen.plot(x = 'ID', y = ['dose', 'len'], subplots = True, title = ['Vitamin C Dosis', 'Länge zahnbildender Zellen'])
```



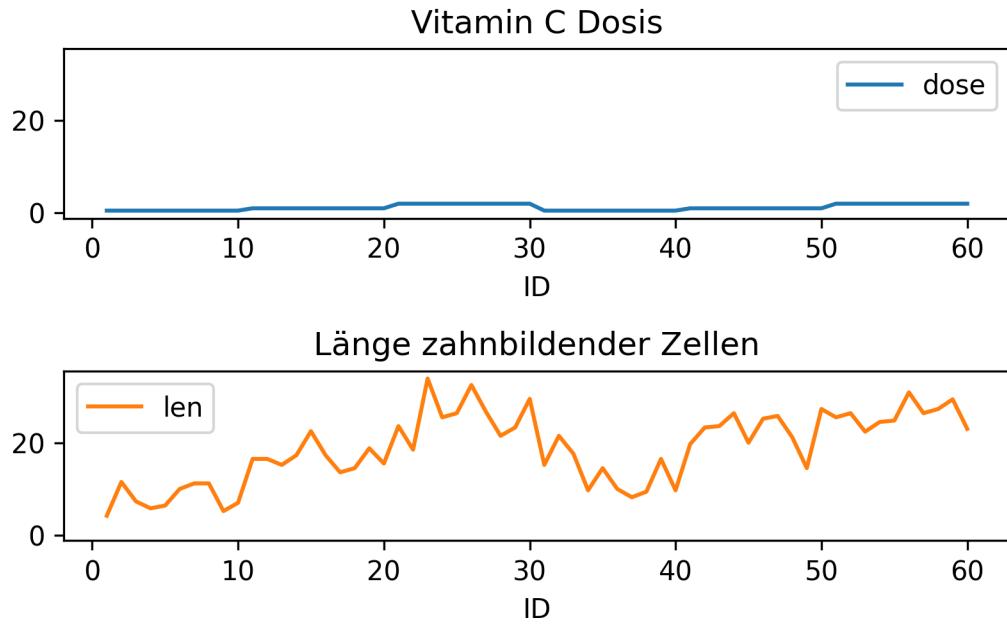
```
meerschweinchen.plot(x = 'ID', y = ['dose', 'len'], subplots = True, sharex = False, sharey = True)
```

Vitamin C Dosis



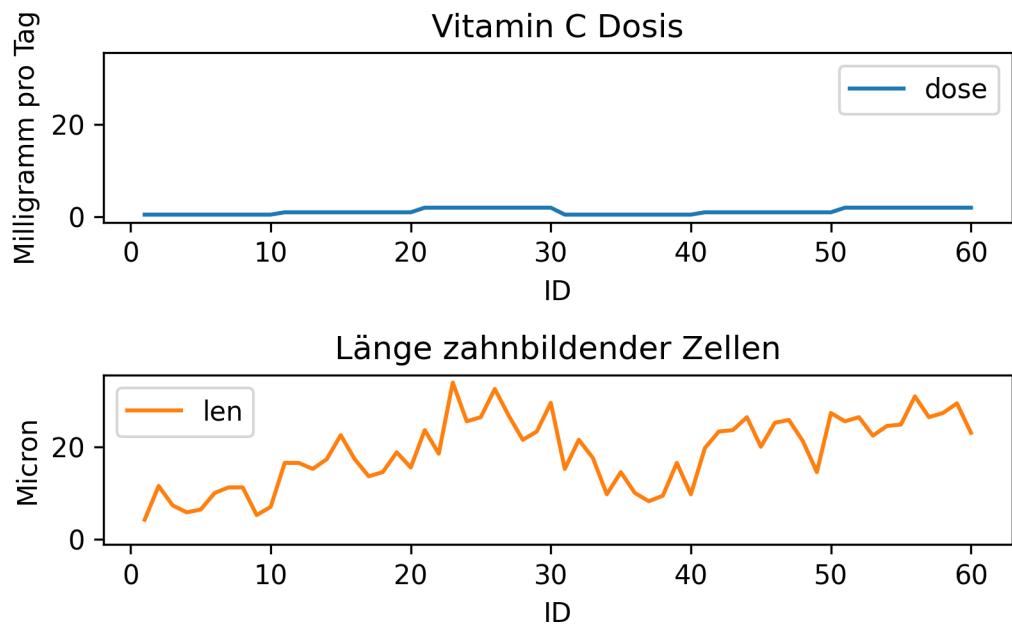
```
import matplotlib.pyplot as plt

meerschweinchen.plot(x = 'ID', y = ['dose', 'len'], subplots = True, sharex = False, sharey =
```



```
mein_plot = meerschweinchen.plot(x = 'ID', y = ['dose', 'len'], subplots = True, sharex = False)

mein_plot[0].set_ylabel('Milligramm pro Tag')
mein_plot[1].set_ylabel('Micron')
plt.tight_layout()
plt.show()
```



## 45 Datentypen

```
skalar = np.array([2])
print(skalar.dtype, "\n")

skalar = np.array([2.1])

print(skalar.dtype)
```

## Byte-Reihenfolge

```
skalar = np.array(['2'])
print(skalar.dtype, "\n")

skalar = np.array(['2.1'])
print(skalar.dtype, "\n")

# Ein Datentyp mit mehr Speicherplatzbedarf kann zugewiesen werden
skalar = np.array([2], dtype = 'U3')
print(skalar.dtype)
```

[NumPy-Dokumentation](#)

[Pandas Dokumentation](#)

- Kategorie
- Zeitzonenbewusstes Datumsformat
- 

```
# NumPy-Datentyp int
series = pd.Series([1, 2, 3], dtype = 'int')
print(series, "\n")

# NumPy-Datentyp int unterstützt fehlende Werte nicht
try:
    series = pd.Series([1, 2, 3, np.nan], dtype = 'int')
```

```
except Exception as error:  
    print(error, "\n")  
  
# Pandas-Datentyp Int64 unterstützt fehlende Werte  
series = pd.Series([1, 2, 3, np.nan], dtype = 'Int64')  
print(series)
```

### ⚠ Warning ??: Pandas-Datentyp string

Pandas nutzt wie die Pythonbasis den Datentyp ‘string’, der unveränderlich (immutable) ist. Das bedeutet, es gibt keine Methode, die eine angelegte Zeichenkette verändern kann. Operationen mit diesem Datentyp geben ein neues Objekt mit dem Datentyp ‘string’ zurück.

Die Übergabe des Datentyps ‘str’ führt zur Verwendung des NumPy-Datentyps string (dtype = ‘str’), der veränderlich (mutable) ist.

Je nach Situation kann die Verwendung des einen oder des anderen Datentyps nützlich sein. Beispielsweise kann der NumPy-Datentyp ‘str’ mit der Methode `pd.Series.sum()` verkettet werden.

```
# mit NumPy-Datentyp 'str'  
string_series = pd.Series(['H', 'a', 'l', 'l', 'o', '!'], dtype = 'str')  
print(f"Mit NumPy-Datentyp 'str': {string_series.sum()}")  
  
# mit Pandas-Datentyp 'string'  
try:  
    string_series.astype('string').sum()  
except Exception as error:  
    print("\nMit Pandas-Datentyp 'string':")  
    print(error)
```

Mit NumPy-Datentyp 'str': Hallo!

Mit Pandas-Datentyp 'string':  
Cannot perform reduction 'sum' with string dtype

# 46 Zeitreihen

- 
- 

## 46.1 Datums- und Zeitinformationen in Python

- Dokumentation des Moduls time
- Dokumentation des Moduls datetime
- Dokumentation des Moduls calendar

Wikipedia  
[pytz](#) Dokumentation

- <https://numpy.org/doc/stable/reference/arrays.datetime.html>
- [https://pandas.pydata.org/docs/user\\_guide/timeseries.html](https://pandas.pydata.org/docs/user_guide/timeseries.html)

#### 46.1.1 Naive und bewusste Datetime-Objekte

“Deprecated since version 1.11.0: NumPy does not store timezone information. For backwards compatibility, datetime64 still parses timezone offsets, which it handles by converting to UTC $\pm$ 00:00 (Zulu time). This behaviour is deprecated and will raise an error in the future.” [NumPy Dokumentation](#)

##### 46.1.1.1 Zeitzonen

```
zeitreihe = pd.Series(pd.date_range(start = "2023-03-26T00:00", end = "2023-03-27T00:00", freq="H"))
zeitreihe
```

```
pd.to_datetime(zeitreihe, utc = True)
```

```
zeitreihe.dt.tz_convert(tz = 'portugal')
```

**i** Note ??: verfügbare Zeitzonen ermitteln

Der folgende Code gibt die in Python verfügbaren Zeitzonen aus.

```
from zoneinfo import available_timezones

for timezone in sorted(available_timezones()):
    print(timezone)

Africa/Abidjan
Africa/Accra
Africa/Addis_Ababa
Africa/Algiers
Africa/Asmara
Africa/Asmera
Africa/Bamako
Africa/Bangui
Africa/Banjul
Africa/Bissau
Africa/Blantyre
Africa/Brazzaville
Africa/Bujumbura
Africa/Cairo
Africa/Casablanca
Africa/Ceuta
Africa/Conakry
Africa/Dakar
Africa/Dar_es_Salaam
Africa/Djibouti
Africa/Douala
Africa/El_Aaiun
Africa/Freetown
Africa/Gaborone
Africa/Harare
Africa/Johannesburg
Africa/Juba
Africa/Kampala
Africa/Khartoum
Africa/Kigali
Africa/Kinshasa
Africa/Lagos
Africa/Libreville
Africa/Lome
Africa/Luanda
Africa/Lubumbashi
```

Africa/Lusaka  
Africa/Malabo  
Africa/Maputo  
Africa/Maseru  
Africa/Mbabane  
Africa/Mogadishu  
Africa/Monrovia  
Africa/Nairobi  
Africa/Ndjamena  
Africa/Niamey  
Africa/Nouakchott  
Africa/Ouagadougou  
Africa/Porto-Novo  
Africa/Sao\_Tome  
Africa/Timbuktu  
Africa/Tripoli  
Africa/Tunis  
Africa/Windhoek  
America/Adak  
America/Anchorage  
America/Anguilla  
America/Antigua  
America/Araguaina  
America/Argentina/Buenos\_Aires  
America/Argentina/Catamarca  
America/Argentina/ComodRivadavia  
America/Argentina/Cordoba  
America/Argentina/Jujuy  
America/Argentina/La\_Rioja  
America/Argentina/Mendoza  
America/Argentina/Rio\_Gallegos  
America/Argentina/Salta  
America/Argentina/San\_Juan  
America/Argentina/San\_Luis  
America/Argentina/Tucuman  
America/Argentina/Ushuaia  
America/Aruba  
America/Asuncion  
America/Atikokan  
America/Atka  
America/Bahia

America/Bahia\_Banderas  
America/Barbados  
America/Belem  
America/Belize  
America/Blanc-Sablon  
America/Boa\_Vista  
America/Bogota  
America/Boise  
America/Buenos\_Aires  
America/Cambridge\_Bay  
America/Campo\_Grande  
America/Cancun  
America/Caracas  
America/Catamarca  
America/Cayenne  
America/Cayman  
America/Chicago  
America/Chihuahua  
America/Ciudad\_Juarez  
America/Coral\_Harbour  
America/Cordoba  
America/Costa\_Rica  
America/Coyhaique  
America/Creston  
America/Cuiaba  
America/Curacao  
America/Danmarkshavn  
America/Dawson  
America/Dawson\_Creek  
America/Denver  
America/Detroit  
America/Dominica  
America/Edmonton  
America/Eirunepe  
America/El\_Salvador  
America/Ensenada  
America/Fort\_Nelson  
America/Fort\_Wayne  
America/Fortaleza  
America/Glace\_Bay  
America/Godthab

America/Goose\_Bay  
America/Grand\_Turk  
America/Grenada  
America/Guadeloupe  
America/Guatemala  
America/Guayaquil  
America/Guyana  
America/Halifax  
America/Havana  
America/Hermosillo  
America/Indiana/Indianapolis  
America/Indiana/Knox  
America/Indiana/Marengo  
America/Indiana/Petersburg  
America/Indiana/Tell\_City  
America/Indiana/Vevay  
America/Indiana/Vincennes  
America/Indiana/Winamac  
America/Indianapolis  
America/Inuvik  
America/Iqaluit  
America/Jamaica  
America/Jujuy  
America/Juneau  
America/Kentucky/Louisville  
America/Kentucky/Monticello  
America/Knox\_IN  
America/Kralendijk  
America/La\_Paz  
America/Lima  
America/Los\_Angeles  
America/Louisville  
America/Lower\_Princes  
America/Maceio  
America/Managua  
America/Manaus  
America/Marigot  
America/Martinique  
America/Matamoros  
America/Mazatlan  
America/Mendoza

America/Menominee  
America/Merida  
America/Metlakatla  
America/Mexico\_City  
America/Miquelon  
America/Moncton  
America/Monterrey  
America/Montevideo  
America/Montreal  
America/Montserrat  
America/Nassau  
America/New\_York  
America/Nipigon  
America/Nome  
America/Noronha  
America/North\_Dakota/Beulah  
America/North\_Dakota/Center  
America/North\_Dakota/New\_Salem  
America/Nuuk  
America/Ojinaga  
America/Panama  
America/Pangnirtung  
America/Paramaribo  
America/Phoenix  
America/Port-au-Prince  
America/Port\_of\_Spain  
America/Porto\_Acre  
America/Porto\_Velho  
America/Puerto\_Rico  
America/Punta\_Arenas  
America/Rainy\_River  
America/Rankin\_Inlet  
America/Recife  
America/Regina  
America/Resolute  
America/Rio\_Branco  
America/Rosario  
America/Santa\_Isabel  
America/Santarem  
America/Santiago  
America/Santo\_Domingo

America/Sao\_Paulo  
America/Scoresbysund  
America/Shiprock  
America/Sitka  
America/St\_Barthelemy  
America/St\_Johns  
America/St\_Kitts  
America/St\_Lucia  
America/St\_Thomas  
America/St\_Vincent  
America/Swift\_Current  
America/Tegucigalpa  
America/Thule  
America/Thunder\_Bay  
America/Tijuana  
America/Toronto  
America/Tortola  
America/Vancouver  
America/Virgin  
America/Whitehorse  
America/Winnipeg  
America/Yakutat  
America/Yellowknife  
Antarctica/Casey  
Antarctica/Davis  
Antarctica/DumontDUrville  
Antarctica/Macquarie  
Antarctica/Mawson  
Antarctica/Mcmurdo  
Antarctica/Palmer  
Antarctica/Rothera  
Antarctica/South\_Pole  
Antarctica/Syowa  
Antarctica/Troll  
Antarctica/Vostok  
Arctic/Longyearbyen  
Asia/Aden  
Asia/Almaty  
Asia/Amman  
Asia/Anadyr  
Asia/Aqtau

Asia/Aqtobe  
Asia/Ashgabat  
Asia/Ashkhabad  
Asia/Atyrau  
Asia/Baghdad  
Asia/Bahrain  
Asia/Baku  
Asia/Bangkok  
Asia/Barnaul  
Asia/Beirut  
Asia/Bishkek  
Asia/Brunei  
Asia/Calcutta  
Asia/Chita  
Asia/Choibalsan  
Asia/Chongqing  
Asia/Chungking  
Asia/Colombo  
Asia/Dacca  
Asia/Damascus  
Asia/Dhaka  
Asia/Dili  
Asia/Dubai  
Asia/Dushanbe  
Asia/Famagusta  
Asia/Gaza  
Asia/Harbin  
Asia/Hebron  
Asia/Ho\_Chi\_Minh  
Asia/Hong\_Kong  
Asia/Hovd  
Asia/Irkutsk  
Asia/Istanbul  
Asia/Jakarta  
Asia/Jayapura  
Asia/Jerusalem  
Asia/Kabul  
Asia/Kamchatka  
Asia/Karachi  
Asia/Kashgar  
Asia/Kathmandu

Asia/Katmandu  
Asia/Khandyga  
Asia/Kolkata  
Asia/Krasnoyarsk  
Asia/Kuala\_Lumpur  
Asia/Kuching  
Asia/Kuwait  
Asia/Macao  
Asia/Macau  
Asia/Magadan  
Asia/Makassar  
Asia/Manila  
Asia/Muscat  
Asia/Nicosia  
Asia/Novokuznetsk  
Asia/Novosibirsk  
Asia/Omsk  
Asia/Oral  
Asia/Phnom\_Penh  
Asia/Pontianak  
Asia/Pyongyang  
Asia/Qatar  
Asia/Qostanay  
Asia/Qyzylorda  
Asia/Rangoon  
Asia/Riyadh  
Asia/Saigon  
Asia/Sakhalin  
Asia/Samarkand  
Asia/Seoul  
Asia/Shanghai  
Asia/Singapore  
Asia/Srednekolymsk  
Asia/Taipei  
Asia/Tashkent  
Asia/Tbilisi  
Asia/Tehran  
Asia/Tel\_Aviv  
Asia/Thimbu  
Asia/Thimphu  
Asia/Tokyo

Asia/Tomsk  
Asia/Ujung\_Pandang  
Asia/Ulaanbaatar  
Asia/Ulan\_Bator  
Asia/Urumqi  
Asia/Ust-Nera  
Asia/Vientiane  
Asia/Vladivostok  
Asia/Yakutsk  
Asia/Yangon  
Asia/Yekaterinburg  
Asia/Yerevan  
Atlantic/Azores  
Atlantic/Bermuda  
Atlantic/Canary  
Atlantic/Cape\_Verde  
Atlantic/Faeroe  
Atlantic/Faroe  
Atlantic/Jan\_Mayen  
Atlantic/Madeira  
Atlantic/Reykjavik  
Atlantic/South\_Georgia  
Atlantic/St\_Helena  
Atlantic/Stanley  
Australia/ACT  
Australia/Adelaide  
Australia/Brisbane  
Australia/Broken\_Hill  
Australia/Canberra  
Australia/Currie  
Australia/Darwin  
Australia/Eucla  
Australia/Hobart  
Australia/LHI  
Australia/Lindeman  
Australia/Lord\_Howe  
Australia/Melbourne  
Australia/NSW  
Australia/North  
Australia/Perth  
Australia/Queensland

Australia/South  
Australia/Sydney  
Australia/Tasmania  
Australia/Victoria  
Australia/West  
Australia/Yancowinna  
Brazil/Acre  
Brazil/DeNoronha  
Brazil/East  
Brazil/West  
CET  
CST6CDT  
Canada/Atlantic  
Canada/Central  
Canada/Eastern  
Canada/Mountain  
Canada/Newfoundland  
Canada/Pacific  
Canada/Saskatchewan  
Canada/Yukon  
Chile/Continental  
Chile/EasterIsland  
Cuba  
EET  
EST  
EST5EDT  
Egypt  
Eire  
Etc/GMT  
Etc/GMT+0  
Etc/GMT+1  
Etc/GMT+10  
Etc/GMT+11  
Etc/GMT+12  
Etc/GMT+2  
Etc/GMT+3  
Etc/GMT+4  
Etc/GMT+5  
Etc/GMT+6  
Etc/GMT+7  
Etc/GMT+8

Etc/GMT+9  
Etc/GMT-0  
Etc/GMT-1  
Etc/GMT-10  
Etc/GMT-11  
Etc/GMT-12  
Etc/GMT-13  
Etc/GMT-14  
Etc/GMT-2  
Etc/GMT-3  
Etc/GMT-4  
Etc/GMT-5  
Etc/GMT-6  
Etc/GMT-7  
Etc/GMT-8  
Etc/GMT-9  
Etc/GMT0  
Etc/Greenwich  
Etc/UCT  
Etc/UTC  
Etc/Universal  
Etc/Zulu  
Europe/Amsterdam  
Europe/Andorra  
Europe/Astrakhan  
Europe/Athens  
Europe/Belfast  
Europe/Belgrade  
Europe/Berlin  
Europe/Bratislava  
Europe/Brussels  
Europe/Bucharest  
Europe/Budapest  
Europe/Busingen  
Europe/Chisinau  
Europe/Copenhagen  
Europe/Dublin  
Europe/Gibraltar  
Europe/Guernsey  
Europe/Helsinki  
Europe/Isle\_of\_Man

Europe/Istanbul  
Europe/Jersey  
Europe/Kaliningrad  
Europe/Kiev  
Europe/Kirov  
Europe/Kyiv  
Europe/Lisbon  
Europe/Ljubljana  
Europe/London  
Europe/Luxembourg  
Europe/Madrid  
Europe/Malta  
Europe/Mariehamn  
Europe/Minsk  
Europe/Monaco  
Europe/Moscow  
Europe/Nicosia  
Europe/Oslo  
Europe/Paris  
Europe/Podgorica  
Europe/Prague  
Europe/Riga  
Europe/Rome  
Europe/Samara  
Europe/San\_Marino  
Europe/Sarajevo  
Europe/Saratov  
Europe/Simferopol  
Europe/Skopje  
Europe/Sofia  
Europe/Stockholm  
Europe/Tallinn  
Europe/Tirane  
Europe/Tiraspol  
Europe/Ulyanovsk  
Europe/Uzhgorod  
Europe/Vaduz  
Europe/Vatican  
Europe/Vienna  
Europe/Vilnius  
Europe/Volgograd

Europe/Warsaw  
Europe/Zagreb  
Europe/Zaporozhye  
Europe/Zurich  
Factory  
GB  
GB-Eire  
GMT  
GMT+0  
GMT-0  
GMTO  
Greenwich  
HST  
Hongkong  
Iceland  
Indian/Antananarivo  
Indian/Chagos  
Indian/Christmas  
Indian/Cocos  
Indian/Comoro  
Indian/Kerguelen  
Indian/Mahe  
Indian/Maldives  
Indian/Mauritius  
Indian/Mayotte  
Indian/Reunion  
Iran  
Israel  
Jamaica  
Japan  
Kwajalein  
Libya  
MET  
MST  
MST7MDT  
Mexico/BajaNorte  
Mexico/BajaSur  
Mexico/General  
NZ  
NZ-CHAT  
Navajo

PRC  
PST8PDT  
Pacific/Apia  
Pacific/Auckland  
Pacific/Bougainville  
Pacific/Chatham  
Pacific/Chuuk  
Pacific/Easter  
Pacific/Efate  
Pacific/Enderbury  
Pacific/Fakaofo  
Pacific/Fiji  
Pacific/Funafuti  
Pacific/Galapagos  
Pacific/Gambier  
Pacific/Guadalcanal  
Pacific/Guam  
Pacific/Honolulu  
Pacific/Johnston  
Pacific/Kanton  
Pacific/Kiritimati  
Pacific/Kosrae  
Pacific/Kwajalein  
Pacific/Majuro  
Pacific/Marquesas  
Pacific/Midway  
Pacific/Nauru  
Pacific/Niue  
Pacific/Norfolk  
Pacific/Noumea  
Pacific/Pago\_Pago  
Pacific/Palau  
Pacific/Pitcairn  
Pacific/Pohnpei  
Pacific/Ponape  
Pacific/Port\_Moresby  
Pacific/Rarotonga  
Pacific/Saipan  
Pacific/Samoa  
Pacific/Tahiti  
Pacific/Tarawa

Pacific/Tongatapu  
Pacific/Truk  
Pacific/Wake  
Pacific/Wallis  
Pacific/Yap  
Poland  
Portugal  
ROC  
ROK  
Singapore  
Turkey  
UCT  
US/Alaska  
US/Aleutian  
US/Arizona  
US/Central  
US/East-Indiana  
US/Eastern  
US/Hawaii  
US/Indiana-Starke  
US/Michigan  
US/Mountain  
US/Pacific  
US/Samoa  
UTC  
Universal  
W-SU  
WET  
Zulu

#### 46.1.2 Alles ist relativ: die Epoche

```
import pandas as pd
print(pd.to_datetime(0))
```

### **⚠ Warning ??: Zeit - atomar, koordiniert oder universal?**

NumPy nutzt die Internationale Atomzeit (abgekürzt TAI für französisch Temps Atomique International). Diese nimmt für jeden Kalendertag eine Länge von 86.400 Sekunden an, kennt also keine Schaltsekunde. Die Atomzeit bildet die Grundlage für die koordinierte Weltzeit UTC.

UTC steht für Coordinated Universal Time (auch bekannt als Greenwich Mean Time). Das Kürzel UTC ist ein Kompromiss für die englische und die französische Sprache. Die koordinierte Weltzeit gleicht die Verlangsamung der Erdrotation (astronomisch gemessen als Universalzeit, Universal Time UT) durch Schaltsekunden aus, um die geringfügige Verlängerung eines Tages auszugleichen. Die TAI geht deshalb gegenüber der UTC vor. Seit 1972 unterscheiden sich beide Zeiten um eine ganzzahlige Anzahl von Sekunden. Aktuell (2024) geht die TAI 37 Sekunden gegenüber UTC vor.

Eine Umwandlung in die koordinierte Weltzeit ist in NumPy bislang noch nicht umgesetzt. ([Dokumentation NumPy](#), [Wikipedia](#)).

#### **46.1.3 Zeitumstellung - Daylight Saving Time**

“DST is Daylight Saving Time, an adjustment of the timezone by (usually) one hour during part of the year. DST rules are magic (determined by local law) and can change from year to year. The C library has a table containing the local rules (often it is read from a system file for flexibility) and is the only source of True Wisdom in this respect.” ([Dokumentation time](#))

```
print("Keine Zeitumstellung in UTC:")
print(pd.Timestamp("2025-03-29T09:00") + pd.Timedelta(24, "h"), "\n")

print("Zeitzone mit Zeitumstellung:")
```

```
print(pd.Timestamp("2025-03-29T09:00", tz="Europe/Berlin") + pd.Timedelta(24, "h"), "\n")

print("Heute keine Zeitumstellung in Türkei:")
print(pd.Timestamp("2025-03-29T09:00", tz="Turkey") + pd.Timedelta(24, "h"), "\n")

print("Türkei vor der Abschaffung der Zeitumstellung:")
print(pd.Timestamp("2014-03-30T09:00", tz="Turkey") + pd.Timedelta(24, "h"))
```

[https://en.wikipedia.org/wiki/List\\_of\\_tz\\_database\\_time\\_zones](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones)

#### 46.1.4 Kalender

proleptischen Gregorianischen Kalender  
astronomischer Jahresnumerierung  
NumPy  
[Dokumentation](#)

### 46.2 datetime in Pandas

- 
- 

```
print(pd.to_datetime(1000, unit = 'D'))
print(pd.to_datetime(1000 * 1000, unit = 'h'))
print(pd.to_datetime(1000 * 1000 * 1000, unit = 's'))
```

- 

#### ISO 8601

```
print(pd.to_datetime('2017'))
print(pd.to_datetime('2017-01-01T00'))
print(pd.to_datetime('2017-01-01 00:00:00'))
```

- 

#### Dokumentation strftime zur string-Formatierung

```
print(pd.to_datetime('Monday, 12. August `24', format = "%A, %d. %B ``%y"))
print(pd.to_datetime('Monday, 12. August 2024, 12:15 Uhr CET', format = "%A, %d. %B %Y, %H:%M"))
```

-

```
print(pd.to_datetime({'year':[2020, 2024], 'month': [1, 11], 'day': [1, 21]}), "\n")
print(pd.to_datetime(pd.DataFrame({'year':[2020, 2024], 'month': [1, 11], 'day': [1, 21]})))
```

- 
- 
- 

Liste verfügbarer strings

```
print(pd.date_range(start = '2017', end = '2024', periods = 3), "\n")
print(pd.date_range(start = '2017', end = '2024', freq = 'Y'), "\n")
print(pd.date_range(end = '2024', freq = 'h', periods = 3))
```

### ⚠ Warning ???: pd.date\_range()

Die Funktion `pd.date_range()` wird künftig das Kürzel 'Y' nicht mehr unterstützen. Stattdessen können die Kürzel 'YS' (Jahresbeginn) oder 'YE' (Jahresende) verwendet werden. Ebenso wird das Kürzel 'M' künftig durch 'MS' (Monatsstart), 'ME' (Monatsende) ersetzt.

## 46.3 timedelta in Pandas

```
print(pd.Timedelta(1, 'D'))
print(pd.Timedelta(days = 1, hours = 1))
```

```
try:
    print(pd.Timedelta(1, 'Y'))
except ValueError as error:
    print(error)
else:
    print(pd.Timedelta(1, 'Y'))
```

```
print(pd.Timedelta('10sec'))
print(pd.Timedelta('10min'))
print(pd.Timedelta('10hours'))
print(pd.Timedelta('10days'))
print(pd.Timedelta('10w'))
```

```
pd.date_range(start = '2024-01-01T00:00', end = '2024-01-01T02:00', freq = '15min') + pd.Time
```

## 46.4 Zugriff auf Zeitreihen

```
# Attribute
print("Jahr:", pd.to_datetime(0).year)
print("Monat:", pd.to_datetime(0).month)
print("Tag:", pd.to_datetime(0).day)
print("Stunde:", pd.to_datetime(0).hour)
print("Minute:", pd.to_datetime(0).minute)
print("Sekunde:", pd.to_datetime(0).second)
print("Tag des Jahres:", pd.to_datetime(0).dayofyear)
print("Wochentag:", pd.to_datetime(0).dayofweek)
print("Tage im Monat:", pd.to_datetime(0).days_in_month)
print("Schaltjahr:", pd.to_datetime(0).is_leap_year)

# Methoden
print("\nDatum:", pd.to_datetime(0).date())
print("Zeit:", pd.to_datetime(0).time())
print("Wochentag (0-6):", pd.to_datetime(0).weekday())
print("Monatsname:", pd.to_datetime(0).month_name())
```

**i** Note ??: Attribute und Methoden eines datetime-Objekts

```
objekt = pd.to_datetime(0)

attribute = [attr for attr in dir(objekt) if not (callable(getattr(0, attr)) or attr.startswith('__'))]
print("Attribute:")
print(30 * "=")
print(attribute)

methoden = [attr for attr in dir(objekt) if (callable(getattr(0, attr)) and not attr.startswith('__'))]
print("\nMethoden:")
print(30 * "=")
print(methoden)
```

### Attribute:

=====  
=====

## Methoden:

=====  
=====

```
['_from_dt64', '_from_value_and_reso', '_round', 'as_unit', 'astimezone', 'ceil', 'combine
```

.dt accessor

## Der dt-Operator

```
# Attribute
print("Datum:", pd.Series(pd.to_datetime(0)).dt.date) # Unterschied
print("Zeit:", pd.Series(pd.to_datetime(0)).dt.time) # Unterschied
print("Jahr", pd.Series(pd.to_datetime(0)).dt.year)
print("Monat", pd.Series(pd.to_datetime(0)).dt.month)
print("Tag", pd.Series(pd.to_datetime(0)).dt.day)
print("Stunde", pd.Series(pd.to_datetime(0)).dt.hour)
print("Minute", pd.Series(pd.to_datetime(0)).dt.minute)
print("Sekunde", pd.Series(pd.to_datetime(0)).dt.second)

print("\nTag des Jahres", pd.Series(pd.to_datetime(0)).dt.dayofyear)
print("Wochentag:", pd.Series(pd.to_datetime(0)).dt.dayofweek)
print("Wochentag:", pd.Series(pd.to_datetime(0)).dt.weekday) # Unterschied
print("Tage im Monat:", pd.Series(pd.to_datetime(0)).dt.days_in_month)
print("Schaltjahr:", pd.Series(pd.to_datetime(0)).dt.is_leap_year)

# Methoden
print("\nName des Monats:", pd.Series(pd.to_datetime(0)).dt.month_name())
```

Datum: 0 1970-01-01  
dtype: object  
Zeit: 0 00:00:00  
dtype: object  
Jahr 0 1970  
dtype: int32  
Monat 0 1  
dtype: int32  
Tag 0 1  
dtype: int32  
Stunde 0 0  
dtype: int32  
Minute 0 0  
dtype: int32  
Sekunde 0 0  
dtype: int32

Tag des Jahres 0 1  
dtype: int32  
Wochentag: 0 3  
dtype: int32

```
Wochentag: 0      3
dtype: int32
Tage im Monat: 0     31
dtype: int32
Schaltjahr: 0    False
dtype: bool

Name des Monats: 0     January
dtype: object
```

## 46.5 Aufgaben

- 1.
- 2.
- 3.
- 4.

💡 Tip ???: Musterlösung

### Aufgabe 1

Ersetzen sie in der Lösung die Zeichenkette ‘YYYY-MM-DD’ bzw., wenn Sie die Uhrzeit Ihrer Geburt kennen, die Zeichenkette ‘YYYY-MM-DDTHH:MM’ durch Ihren Geburtstag.

In Pandas werden die Schlüsselwörter `pd.to_datetime('today')` und `pd.to_datetime('now')` in Nanosekunden aufgelöst.

```
print((pd.to_datetime('today') - pd.to_datetime('YYYY-MM-DD')).days)
print(pd.to_datetime('now') - pd.to_datetime('YYYY-MM-DDTHH:MM')).total_seconds()
```

### Aufgabe 2

```
print(pd.to_datetime('YYYY-MM-DD').day_of_week)
```

### Aufgabe 3

```
(pd.to_datetime('2025-12-25') - pd.to_datetime('now')).days
```

### Aufgabe 4

```
schaltjahre = pd.date_range(start = '1901', end = '2000', freq = 'YE')
schaltjahre = schaltjahre[schaltjahre.is_leap_year]
print(schaltjahre.year)

Index([1904, 1908, 1912, 1916, 1920, 1924, 1928, 1932, 1936, 1940, 1944, 1948,
       1952, 1956, 1960, 1964, 1968, 1972, 1976, 1980, 1984, 1988, 1992, 1996],
      dtype='int32')
```

## **47 Dateien lesen und schreiben**

Pandas Dokumentation

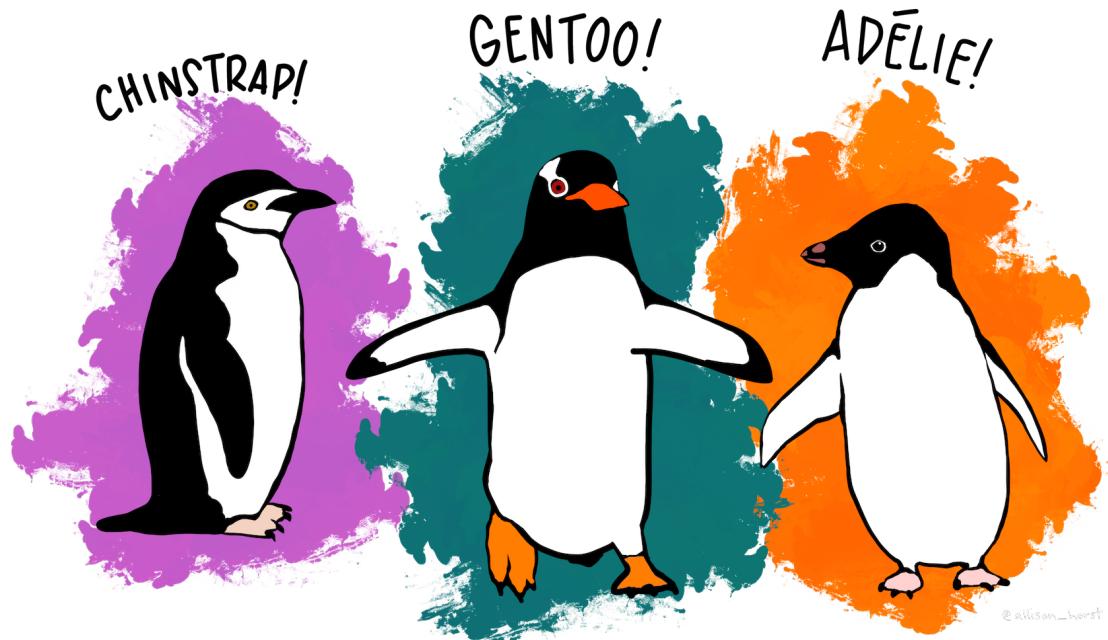


Figure 47.1: Pinguine des Palmer-Station-Datensatzes

CC0-1.0

[GitHub](#)

CCO

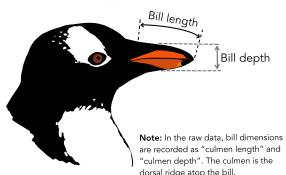
[GitHub](#)

```
# R Befehle, um den Datensatz zu laden
install.packages("palmerpenguins")
library(palmerpenguins)
```

<https://allisonhorst.github.io/palmerpenguins/>

```
penguins = pd.read_csv(filepath_or_buffer = '01-daten/penguins.csv')
```

```
print(penguins.head())
```



CC0-1.0

[GitHub](#)

```
print(penguins.info())
```

```
penguins = pd.read_csv(filepath_or_buffer = '01-daten/penguins.csv', dtype = {'species': 'cat  
# nachträglich  
# penguins = penguins.astype({'species': 'category', 'island': 'category', 'sex': 'category'})  
print(penguins.info())
```

•  
•  
•  
•  
•

```
# Fehlende Werte bestimmen
print(penguins.apply(pd.isna).head(), "\n")

# zeilenweise aggregieren
print(penguins.apply(pd.isna).any(axis = 1).head(), "\n")

# Anzahl der Zeilen mit fehlenden Werten
print(f"Für {penguins.apply(pd.isna).any(axis = 1).sum()} Pinguine liegen unvollständige Werte vor")

# Indexpositionen bestimmen
print(np.where(penguins.apply(pd.isna).any(axis = 1))[0])

# Zeilen entfernen
penguins.drop(np.where(penguins.apply(pd.isna).any(axis = 1))[0], inplace = True)
```

```
print(penguins.info())
```

## 47.1 Zeitreihen einlesen

•  
•

•

ISO 8601

Dokumentation

strftime-

## 47.2 Aufgaben Zeitreihen einlesen

```
stock = pd.read_csv(filepath_or_buffer = '01-daten/Microsoft_Stock.csv')

print(stock.head(), "\n")
print(stock.info())
```

1.

2.

💡 Tip ???: Musterlösung Zeitreihen einlesen

### 1. Aufgabe

```
stock = pd.read_csv(filepath_or_buffer = '01-daten/Microsoft_Stock.csv',
                    parse_dates = ['Date'], # alternativ: [0]
                    date_format = '%m/%d/%Y %H:%M:%S')

print(stock.head(), "\n")
print(stock.info())
```

	Date	Open	High	Low	Close	Volume
0	2015-04-01 16:00:00	40.60	40.76	40.31	40.72	36865322
1	2015-04-02 16:00:00	40.66	40.74	40.12	40.29	37487476
2	2015-04-06 16:00:00	40.34	41.78	40.18	41.55	39223692
3	2015-04-07 16:00:00	41.61	41.91	41.31	41.53	28809375
4	2015-04-08 16:00:00	41.48	41.69	41.04	41.42	24753438

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1511 entries, 0 to 1510
Data columns (total 6 columns):
```

```

#   Column  Non-Null Count Dtype  
----  -----  -----  ----- 
0    Date    1511 non-null   datetime64[ns] 
1    Open    1511 non-null   float64  
2    High    1511 non-null   float64  
3    Low     1511 non-null   float64  
4    Close   1511 non-null   float64  
5    Volume  1511 non-null   int64  
dtypes: datetime64[ns](1), float64(4), int64(1) 
memory usage: 71.0 KB 
None

```

## 2. Aufgabe

Die Pandas-Methode `Series.dt.weekofyear()` wird seit einiger Zeit nicht mehr unterstützt ([siehe Dokumentation](#)). Die Funktion wurde durch `Series.dt.isocalendar().week` ersetzt.

```

# Jahr und Woche isolieren
print(stock['Date'].dt.isocalendar().week.head(), "\n")
print(stock['Date'].dt.isocalendar().year.tail())

# Jahr und Woche in den DataFrame einfügen
stock.insert(loc = 1, column = 'week', value = stock['Date'].dt.isocalendar().week)
stock.insert(loc = 1, column = 'year', value = stock['Date'].dt.isocalendar().year)

# Maximum für jede Woche mit groupby bestimmen
print(stock.groupby(by = ['year', 'week'])['High'].max())

# grafisch darstellen
stock.groupby(by = ['year', 'week'])['High'].max().plot(ylabel = 'Wochenhöchstkurs (intraday)')

0      14
1      14
2      15
3      15
4      15
Name: week, dtype: UInt32

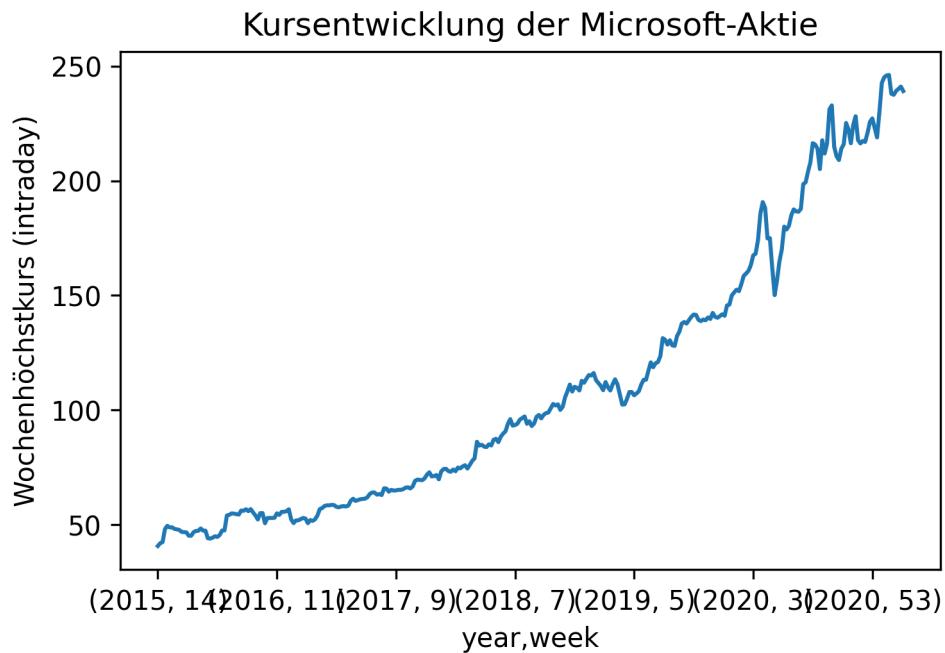
1506    2021
1507    2021
1508    2021

```

```

1509      2021
1510      2021
Name: year, dtype: UInt32
year    week
2015    14      40.76
        15      41.95
        16      42.46
        17      48.14
        18      49.54
        ...
2021    9       237.47
        10      239.17
        11      240.06
        12      241.05
        13      239.10
Name: High, Length: 314, dtype: float64

```



## 47.3 Schwierige Dateien einlesen

[Methodenbaustein Einlesen strukturierter Datensätze](#)



# **Part VII**

## **m-numerik**

# Preamble



Bausteine Computergestützter Datenanalyse. "Numerik" von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Maik Poetzsch und Sebastian Seipel ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar unter <https://github.com/bausteine-der-datenanalyse/m-numerik>. Ausgenommen von der Lizenz sind alle Logos und anders gekennzeichneten Inhalte. 2024

<https://github.com/bausteine-der-datenanalyse/m-numerik>

# **Intro**

## **Voraussetzungen**

- 
- 
- 

## **Verwendete Pakete und Datensätze**

### **Pakete**

- NumPy
- Matplotlib

### **Datensätze**

### **Bearbeitungszeit**

### **Lernziele**

- 
- 
- 
-

# 48 Integration

•  
•  
•

## 48.0.1 Ober- und Untersumme

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Ober- und  
Untersumme

## 48.1 Definition

## 48.2 Beispiel

```
def fkt(x):
    return np.sin(3*x) + 2*x

# Daten für die Visualisierung
x = np.linspace(0, 2, 100)
y = fkt(x)

# Exakte Lösung
I_exakt = (-1/3*np.cos(3*2) + 2**2) - (-1/3)
```

```
n = 5

xi = np.linspace(0, 2, n)
yi = fkt(xi)
```

```
oben = np.zeros(n)
unten = np.zeros(n)

for i in range(len(oben)-1):
    cx = np.linspace(xi[i], xi[i+1], 50)
    cy = fkt(cx)
    oben[i+1] = np.max(cy)
    unten[i+1] = np.min(cy)
```

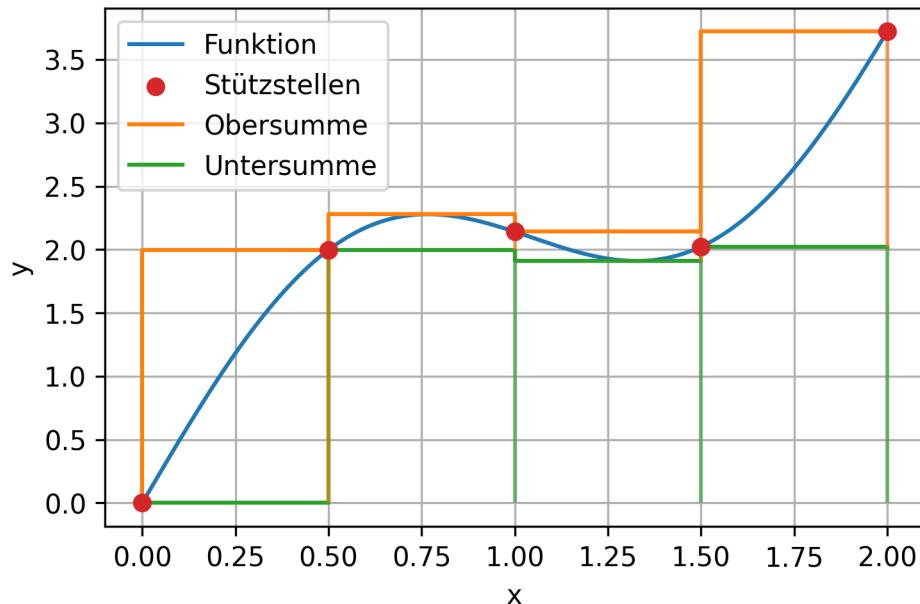
```
oben[0] = yi[0]
unten[0] = yi[0]
```

```
plt.plot(x, y, label='Funktion')
plt.scatter(xi, yi, label='Stützstellen', c='C3', zorder=3)
plt.plot(xi, oben, drawstyle='steps-pre', label='Obersumme')
plt.plot(xi, unten, drawstyle='steps-pre', label='Untersumme')

plt.vlines(xi, ymin=unten, ymax=oben, color='C1', alpha=0.6)
plt.vlines(xi, ymin=0, ymax=unten, color='C2', alpha=0.6)

plt.xlabel('x')
plt.ylabel('y')

plt.grid()
plt.legend();
```



```

def ou_summe(n, a=0, b=2):
    xi = np.linspace(a, b, n)
    yi = fkt(xi)
    dx = xi[1] - xi[0]

    sum_oben = 0
    sum_unten = 0

    for i in range(n-1):
        cx = np.linspace(xi[i], xi[i+1], 50)
        cy = fkt(cx)
        oben = np.max(cy)
        unten = np.min(cy)
        sum_oben += dx * oben
        sum_unten += dx * unten

    return sum_oben, sum_unten

```

```

n_max = 100
ns = np.arange(2, n_max, 1, dtype=int)
os = np.zeros(len(ns))
us = np.zeros(len(ns))

for i, n in enumerate(ns):
    o, u = ou_summe(n)
    os[i] = o
    us[i] = u

```

```

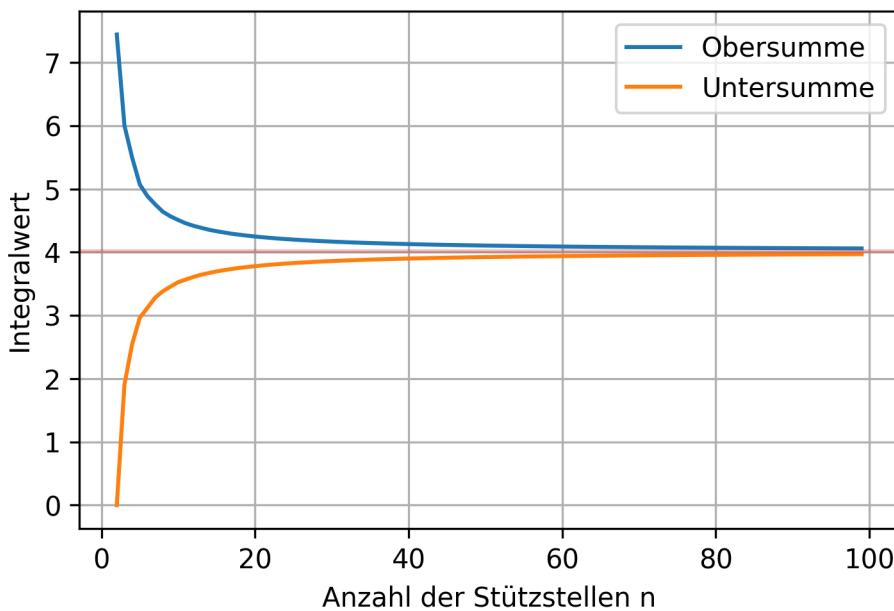
plt.plot(ns, os, label='Obersumme')
plt.plot(ns, us, label='Untersumme')

plt.axhline(y=I_exakt, color='C3', alpha=0.3)

plt.xlabel('Anzahl der Stützstellen n')
plt.ylabel('Integralwert')

plt.grid()
plt.legend();

```

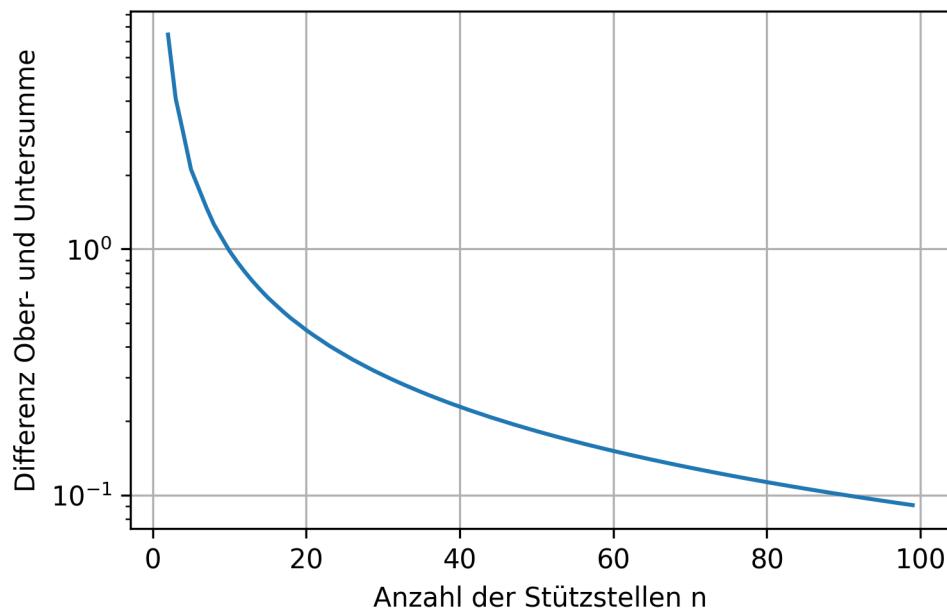


```
plt.plot(ns, os-us)

plt.xlabel('Anzahl der Stützstellen n')
plt.ylabel('Differenz Ober- und Untersumme')

# plt.xscale('log')
# plt.yscale('log')

plt.grid();
```



### 48.3 Interpolation

### 48.3.1 Trapezregel

```
def fkt(x):
    return np.sin(3*x) + 2*x

# Daten für die Visualisierung
x = np.linspace(0, 2, 100)
y = fkt(x)

# Exakte Lösung
I_exakt = (-1/3*np.cos(3*2) + 2**2) - (-1/3)
```

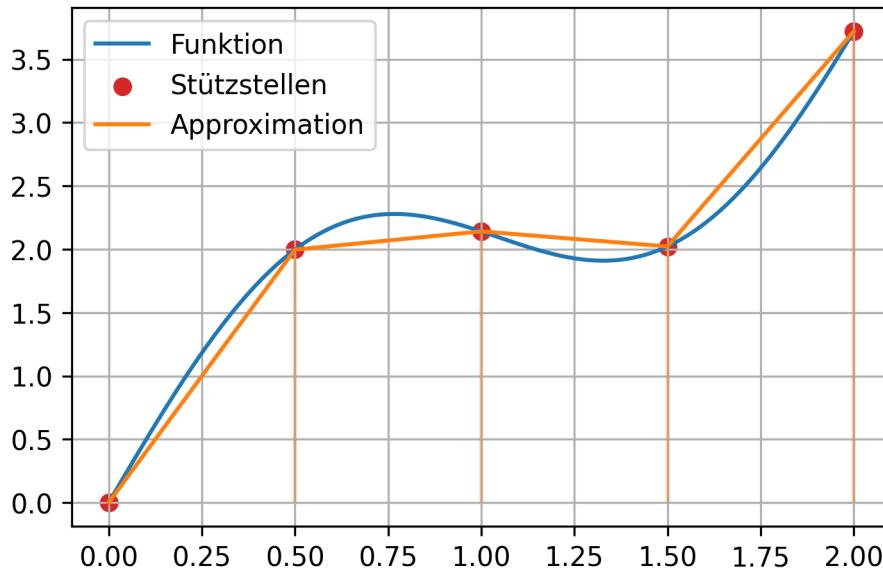
```
n = 5

xi = np.linspace(0, 2, n)
yi = fkt(xi)
```

```
plt.plot(x, y, label='Funktion')
plt.scatter(xi, yi, label='Stützstellen', c='C3')
plt.plot(xi, yi, label='Approximation', c='C1')

plt.vlines(xi, ymin=0, ymax=yi, color='C1', alpha=0.3)

plt.grid()
plt.legend();
```



Funktion `scipy.integrate.trapezoid`

```
res = scipy.integrate.trapezoid(yi, xi)
print(f"Integralwert mit {n} Stützstellen: {res:.4f}")
```

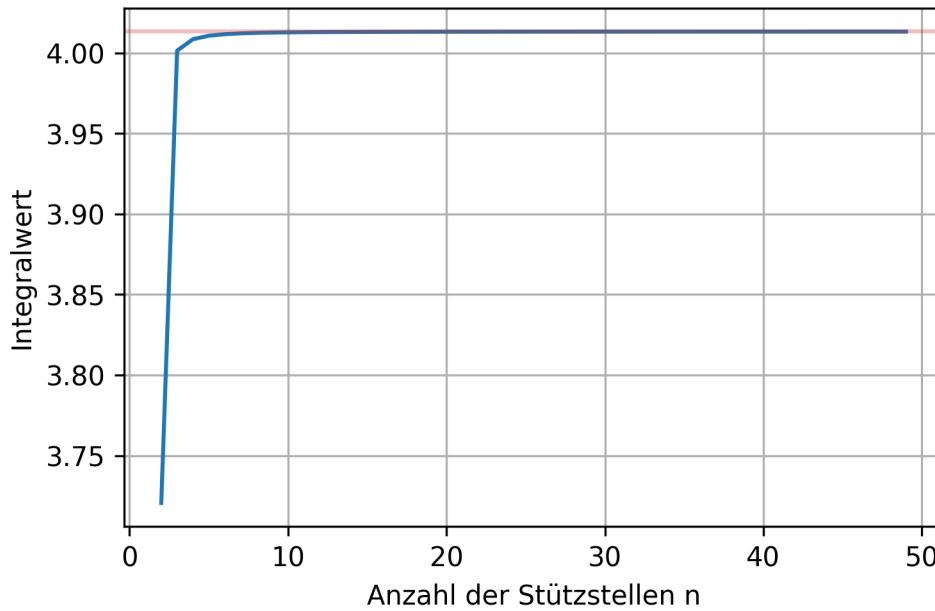
```
n_max = 50
ns = np.arange(2, n_max, 1, dtype=int)
tr = np.zeros(len(ns))

for i, n in enumerate(ns):
    xi = np.linspace(0, 2, n)
    yi = fkt(xi)
    tr[i] = scipy.integrate.trapezoid(yi, xi)
```

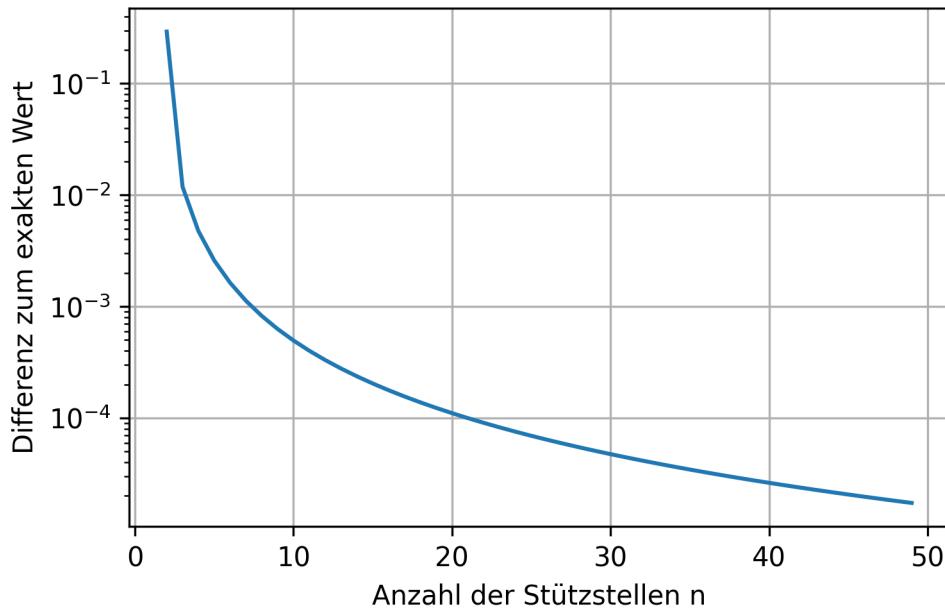
```
plt.plot(ns, tr)
plt.axhline(y=I_exakt, color='C3', alpha=0.3)

plt.xlabel('Anzahl der Stützstellen n')
```

```
plt.ylabel('Integralwert')  
plt.grid();
```



```
plt.plot(ns, np.abs(tr-I_exakt))  
  
plt.xlabel('Anzahl der Stützstellen n')  
plt.ylabel('Differenz zum exakten Wert')  
  
# plt.xscale('log')  
plt.yscale('log')  
  
plt.grid();
```



#### 48.3.2 Simpsonregel

```
n = 5
```

```
xi = np.linspace(0, 2, n)
yi = fkt(xi)
```

```
plt.plot(x, y, label='Funktion')
plt.scatter(xi, yi, label='Stützstellen', c='C3')

# Bestimmung und Plotten der Polynome
for i in range(n-1):
    dx = xi[i+1] - xi[i]
    cx = (xi[i] + xi[i+1]) / 2
    cy = fkt(cx)

P = np.polyfit([xi[i], cx, xi[i+1]], [yi[i], cy, yi[i+1]], 2)
```

```

Px = np.linspace(xi[i], xi[i+1], 20)
Py = np.polyval(P, Px)

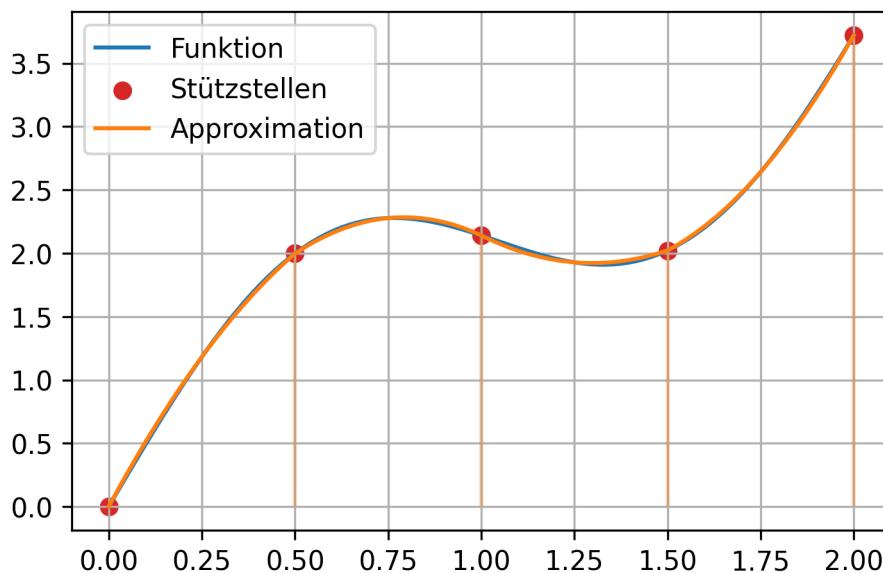
label=None
if i==0:
    label='Approximation'

plt.plot(Px, Py, color='C1', label=label)

plt.vlines(xi, ymin=0, ymax=yi, color='C1', alpha=0.3)

plt.grid()
plt.legend();

```



Funktion `scipy.integrate.simpson`

```

n_max = 50
ns = np.arange(3, n_max, 2, dtype=int)
si = np.zeros(len(ns))
tr = np.zeros(len(ns))

for i, n in enumerate(ns):
    xi = np.linspace(0, 2, n)

```

```

yi = fkt(xi)
si[i] = scipy.integrate.simpson(yi, xi)
tr[i] = scipy.integrate.trapezoid(yi, xi)

```

```

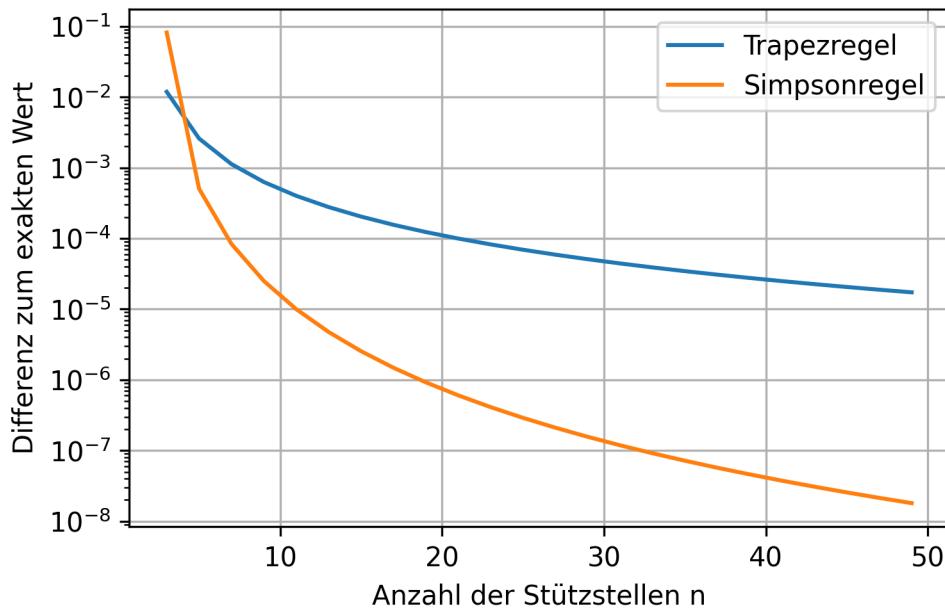
plt.plot(ns, np.abs(tr-I_exakt), label='Trapezregel')
plt.plot(ns, np.abs(si-I_exakt), label='Simpsonregel')

plt.xlabel('Anzahl der Stützstellen n')
plt.ylabel('Differenz zum exakten Wert')

# plt.xscale('log')
plt.yscale('log')

plt.legend()
plt.grid();

```



## 48.4 Monte-Carlo

[Monte-Carlo-Ansatz](#)

```

def fkt(x):
    return np.sin(3*x) + 2*x

# Daten für die Visualisierung
x = np.linspace(0, 2, 100)
y = fkt(x)

# Exakte Lösung
I_exakt = (-1/3*np.cos(3*2) + 2**2) - (-1/3)

```

```

n = 2000
xi = np.random.random(n) * 2
yi = fkt(xi)
I = 2 * 1/n * np.sum(yi)
print(f"Integralwert für {n} Stützstellen: {I:.4f}")

```

```

n_max = 50000
dn = 250
ns = np.arange(dn, n_max, dn, dtype=int)
mc = np.zeros(len(ns))

xi = np.zeros(n_max)

for i, n in enumerate(ns):
    xi[n-dn:n] = np.random.random(dn) * 2
    yi = fkt(xi[:n])
    mc[i] = 2 * 1/n * np.sum(yi)

```

```

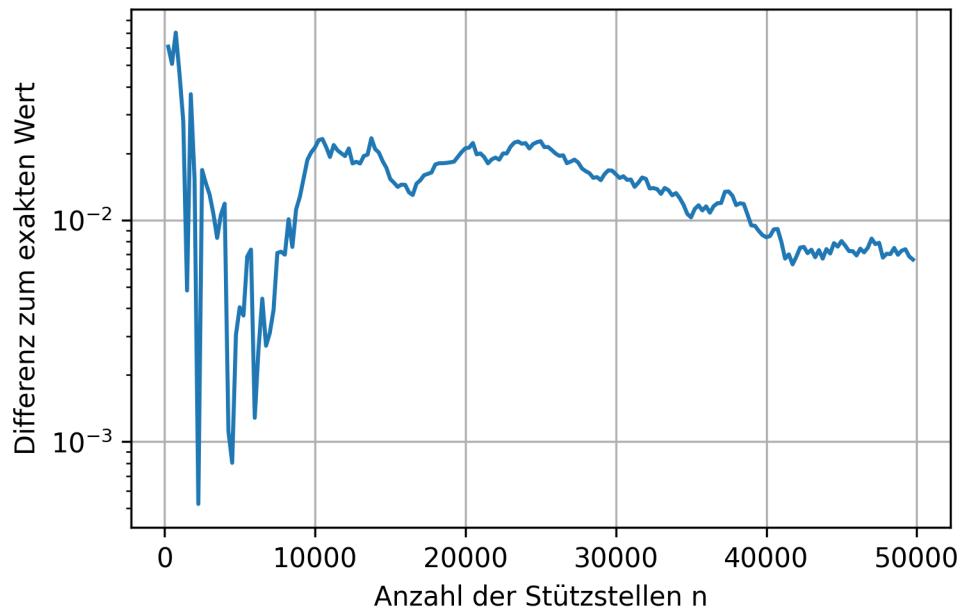
plt.plot(ns, np.abs(mc-I_exakt))

plt.xlabel('Anzahl der Stützstellen n')
plt.ylabel('Differenz zum exakten Wert')

# plt.xscale('log')
# plt.yscale('log')

plt.grid();

```



```

n = 2000
xi = np.random.random(n) * 2
yi = np.random.random(n) * 4

```

```

z = np.sum(yi < fkt(xi))

I = z / n * 8
print(f"Integralwert für {n} Stützstellen: {I}")

```

```

n_max = 50000
dn = 250
ns = np.arange(dn, n_max, dn, dtype=int)
mc = np.zeros(len(ns))

xi = np.zeros(n_max)
yi = np.zeros(n_max)

for i, n in enumerate(ns):
    xi[n-dn:n] = np.random.random(dn) * 2
    yi[n-dn:n] = np.random.random(dn) * 4
    z = np.sum(yi[:n] < fkt(xi[:n]))
    mc[i] = z / n * 8

```

```

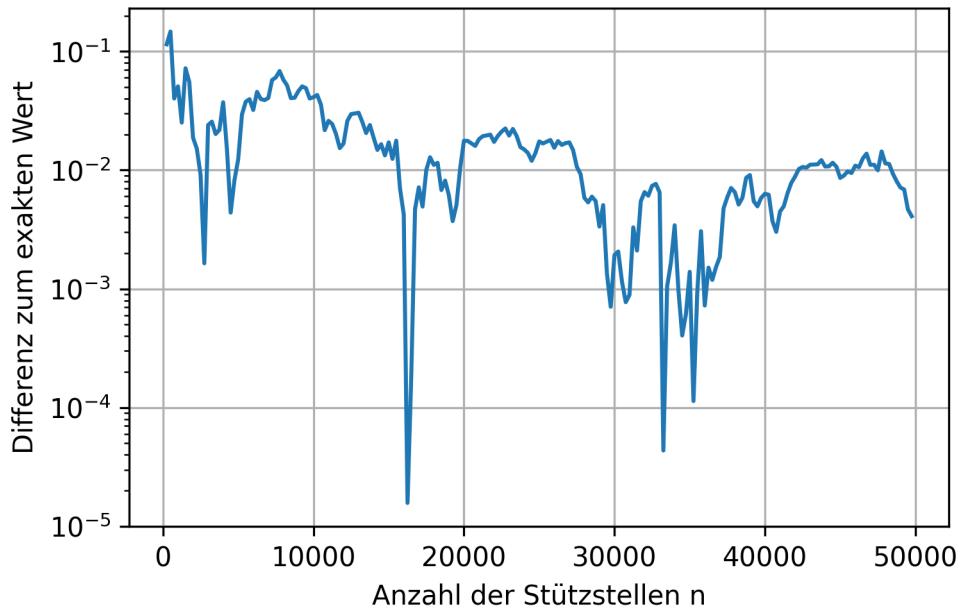
plt.plot(ns, np.abs(mc-I_exakt))

plt.xlabel('Anzahl der Stützstellen n')
plt.ylabel('Differenz zum exakten Wert')

# plt.xscale('log')
# plt.yscale('log')

plt.grid();

```



# 49 Differentiation

## 49.1 Taylor-Entwicklung

Taylor-Entwicklung

```

def fkt(x, p=0):
    if p==0:
        return np.sin(3*x) + 2*x
    if p==1:
        return 3*np.cos(3*x) + 2
    if p==2:
        return -9*np.sin(3*x)
    if p==3:
        return -27*np.cos(3*x)
    return None

# Daten für die Visualisierung
x = np.linspace(0, 2, 100)
y = fkt(x, p=0)

```

```

x0 = 0.85

# Taylor-Elemente
te = []
te.append(0*(x-x0) + fkt(x0, p=0))
te.append((x-x0) * fkt(x0, p=1))
te.append((x-x0)**2 * fkt(x0, p=2) * 1/2)
te.append((x-x0)**3 * fkt(x0, p=3) * 1/6)

```

```

plt.plot(x, y, color='Grey', lw=3, label="Funktion")
plt.plot(x, te[0], label="$\mathsf{\mathcal{O}(1)}$")
plt.plot(x, te[0] + te[1], label="$\mathsf{\mathcal{O}(h)}$")
plt.plot(x, te[0] + te[1] + te[2], label="$\mathsf{\mathcal{O}(h^2)}$")
plt.plot(x, te[0] + te[1] + te[2] + te[3], label="$\mathsf{\mathcal{O}(h^3)}$")

plt.vlines(x0, ymin=0, ymax=fkt(x0), color='Grey', ls='--', alpha=0.5)

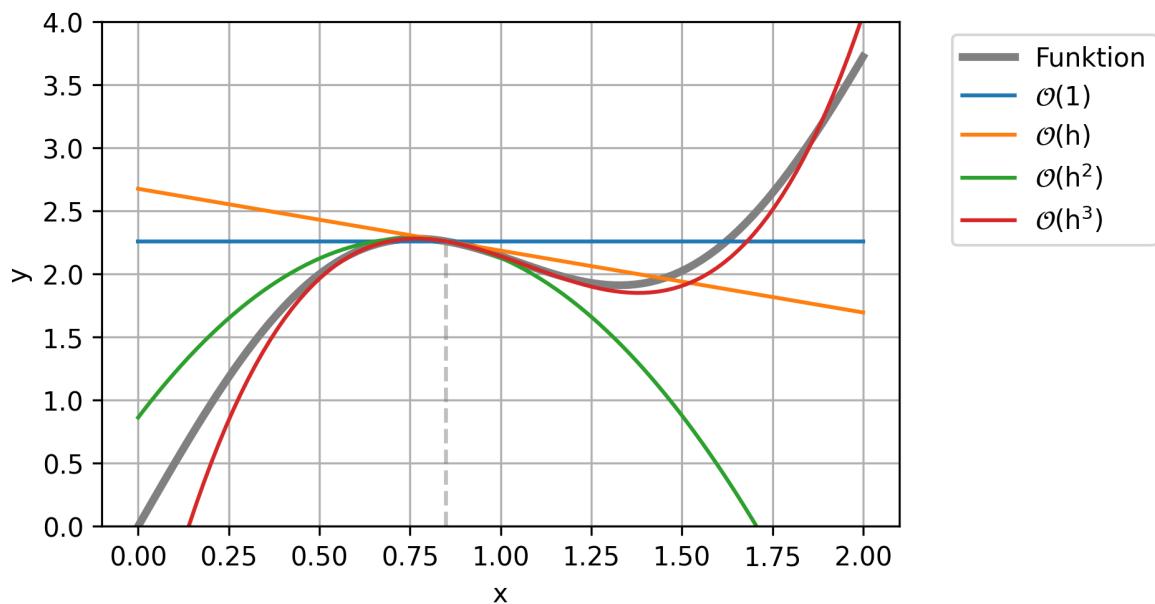
```

```

plt.ylim([0,4])

plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left')
plt.grid()
plt.xlabel('x')
plt.ylabel('y');

```



## 49.2 Differenzenformeln

### 49.2.1 Erste Ableitung erster Ordnung

```
def fkt(x):
    return np.sin(3*x) + 2*x

# Daten für die Visualisierung
x = np.linspace(0, 2, 100)
y = fkt(x)

# Exakte Lösung bei x=0.85
fp_exakt = 3*np.cos(3*0.85) + 2
```

```

# Entwicklungspunkt und Schrittweite
h = 0.25
x0 = 0.85

# Auswertung an den beiden Stellen
f0 = fkt(x0)
fh = fkt(x0 + h)

# Bestimmung der Ableitungsnäherung
fp = (fh - f0) / h

print(f"Die numerische Näherung der Ableitung an der Stelle {x0:.2f}::")
print(f"Näherung mit Schrittweite {h:.2f}: {fp:.2f}")
print(f"Exakter Wert: {fp_exakt:.2f}")

```

```

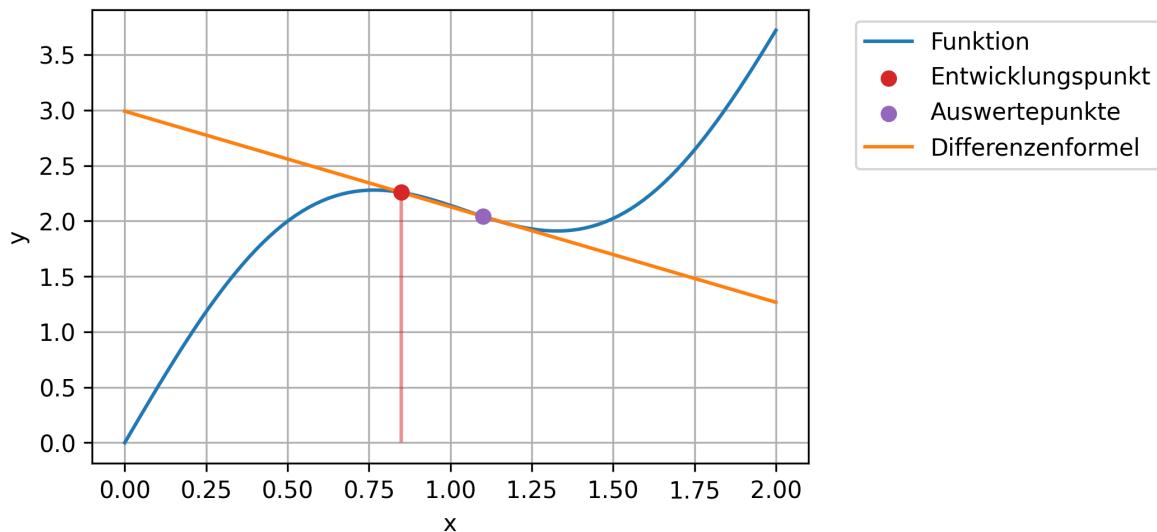
plt.plot(x, y, label="Funktion")
plt.scatter([x0], [f0], color='C3', label='Entwicklungspunkt', zorder=3)
plt.scatter([x0+h], [fh], color='C4', label='Auswertepunkte', zorder=3)

plt.vlines(x0, ymin=0, ymax=f0, color='C3', alpha=0.5)

plt.plot(x, f0 + fp*(x-x0), label='Differenzenformel')

plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left');

```



#### 49.2.2 Erste Ableitung zweiter Ordnung

```

# Auswertung an den beiden Stellen
fnh = fkt(x0 - h)
fph = fkt(x0 + h)

# Bestimmung der Ableitungsnäherung
fp = (fph - fnh) / (2*h)

print(f"Die numerische Näherung der Ableitung an der Stelle {x0:.2f}:")
print(f"Näherung mit Schrittweite {h:.2f}: {fp:.2f}")
print(f"Exakter Wert: {fp_exakt:.2f}")

```

```

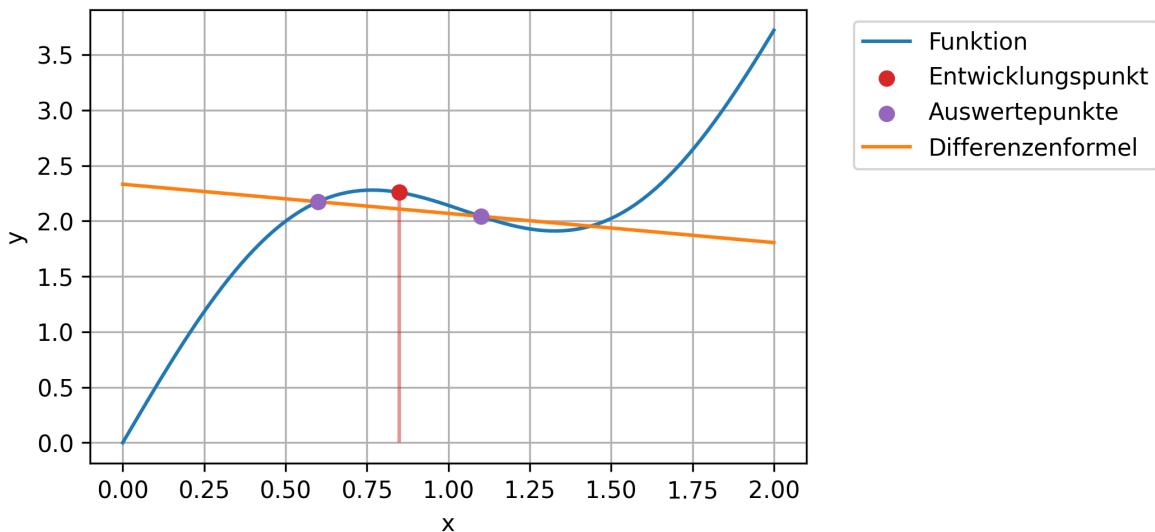
plt.plot(x, y, label="Funktion")
plt.scatter([x0], [f0], color='C3', label='Entwickelpunkt', zorder=3)
plt.scatter([x0-h, x0+h], [fnh, fph], color='C4', label='Auswertepunkte', zorder=3)

plt.vlines(x0, ymin=0, ymax=f0, color='C3', alpha=0.5)

plt.plot(x, fnh + fp*(x-x0+h), label='Differenzenformel')

plt.xlabel('x')
plt.ylabel('y')
plt.grid()
plt.legend(bbox_to_anchor=(1.05, 1.0), loc='upper left');

```



### 49.3 Zweite Ableitung zweiter Ordnung

### 49.4 Fehlerbetrachtung

#### 49.4.1 Approximationsfehler

```
def fkt(x):
    return np.sin(3*x) + 2*x

# Daten für die Visualisierung
```

```

x = np.linspace(0, 2, 100)
y = fkt(x)

# Exakte Lösung bei x=1
fp_exakt = 3*np.cos(3*0.85) + 2

x0 = 0.85

hs = []
fpfs = []
fpcs = []

h0 = 1
for i in range(18):
    h = h0 / 2**i

    f0 = fkt(x0)
    fnh = fkt(x0 - h)
    fph = fkt(x0 + h)

    fpf = (fph - f0) / h
    fpc = (fph - fnh) / (2*h)

    hs.append(h)
    fpfs.append(fpf)
    fpcs.append(fpc)

plt.plot(hs, np.abs(fpfs - fp_exakt), label='vorwärts')
plt.plot(hs, np.abs(fpcs - fp_exakt), label='zentral')

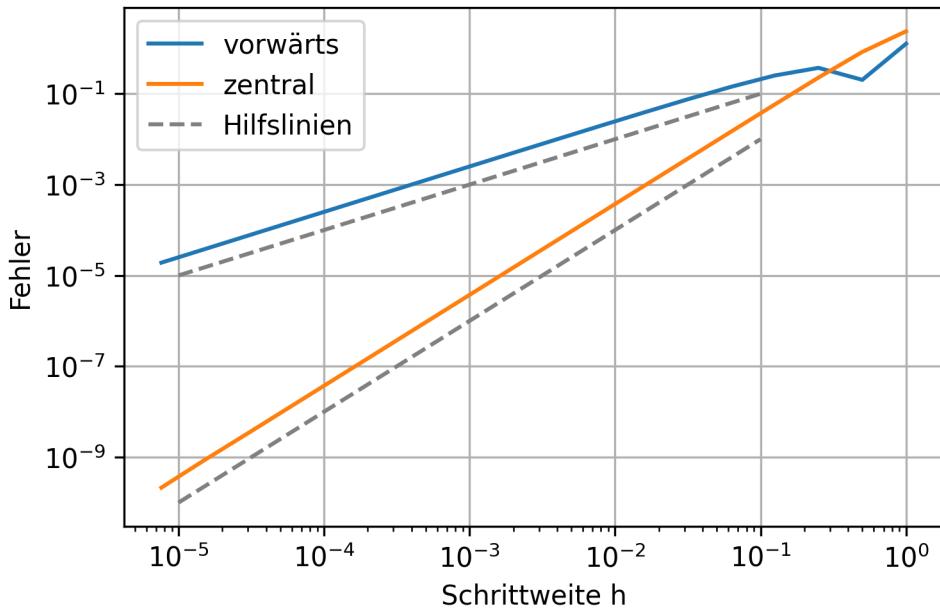
plt.plot([1e-5, 1e-1], [1e-5, 1e-1], '--', color='grey', label='Hilfslinien')
plt.plot([1e-5, 1e-1], [1e-10, 1e-2], '--', color='grey')

plt.xlabel('Schrittweite h')
plt.ylabel('Fehler')

plt.xscale('log')
plt.yscale('log')

plt.legend()
plt.grid();

```



#### 49.4.2 Rundungsfehler

```

x0 = 0.85

hs = []
fpfs = []
fpcs = []

h0 = 1
for i in range(35):
    h = h0 / 2**i

    f0 = fkt(x0)

```

```

fnh = fkt(x0 - h)
fph = fkt(x0 + h)

fpf = (fph - f0) / h
fpc = (fph - fnh) / (2*h)

hs.append(h)
fpfs.append(fpf)
fpcs.append(fpc)

```

```

plt.plot(hs, np.abs(fpfs - fp_exakt), label='vorwärts')
plt.plot(hs, np.abs(fpcs - fp_exakt), label='zentral')

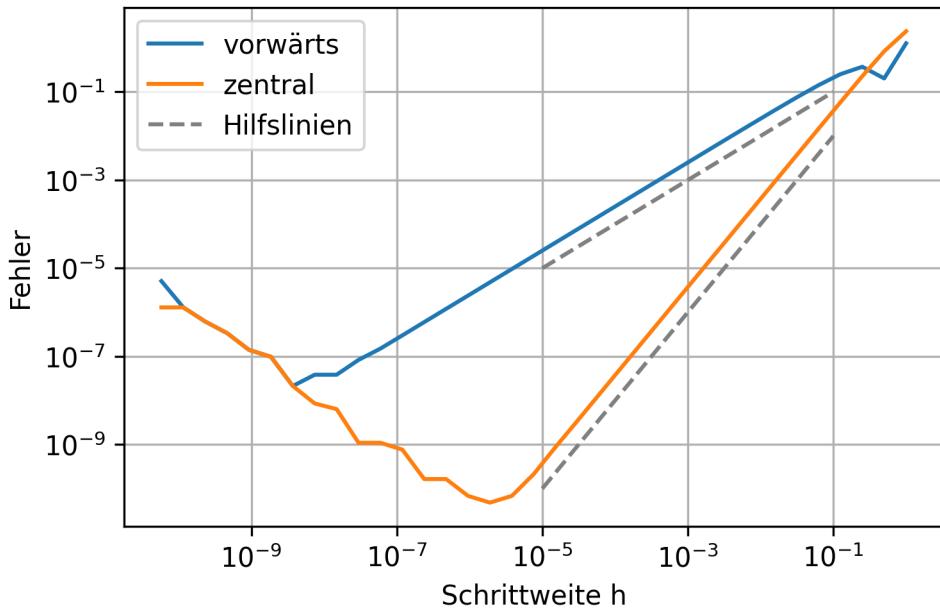
plt.plot([1e-5, 1e-1], [1e-5, 1e-1], '--', color='grey', label='Hilfslinien')
plt.plot([1e-5, 1e-1], [1e-10, 1e-2], '--', color='grey')

plt.xlabel('Schrittweite h')
plt.ylabel('Fehler')

plt.xscale('log')
plt.yscale('log')

plt.legend()
plt.grid();

```



## **Part VIII**

# **m-einlesen-strukturierter-datensätze**

# Methodenbaustein Einlesen strukturierter Datensätze



Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch und Sebastian Seipel. Methodenbaustein Einlesen strukturierter Datensätze von Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar auf [GitHub](#). Ausgenommen von der Lizenz sind alle Logos Dritter und anders gekennzeichneten Inhalte. 2025

[https://github.com/bausteine-  
der-datenanalyse/m-einlesen-strukturierter-datensaetze](https://github.com/bausteine-der-datenanalyse/m-einlesen-strukturierter-datensaetze)

# Voraussetzungen

- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- CSV-Datei
- hier verfügbar
- hier verfügbar
- Datei
- XLS-
- XLS-Datei

- statistischen Ämter des Bundes und der Länder
- XLSX-Datei
- Statistischen Bundesamt
- kostenlose Registrierung bei NASA Earth erforderlich
- kostenlose Registrierung bei NASA Earth erforderlich

## **Lernziele**

- 
- 
- 
-

# 50 Einleitung

heise online

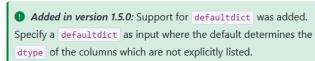
“Unfortunately, the data set is usually dirty, composed of many tables, and has unknown properties. Before any results can be produced, the data must be cleaned and explored—often a long and difficult task. [...] In our experience, the tasks of exploratory data mining and data cleaning constitute 80% of the effort that determines 80% of the value of the ultimate data mining results.” (@Dasu-Johnson-2003, S. ix)

 Tip ??

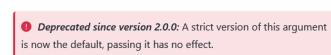
**Schauen Sie sich Ihre Daten an, bevor Sie diese mit Python einlesen!** Dafür reicht ein Texteditor oder ein Tabellenkalkulationsprogramm (hier die automatische Erkennung und Umwandlung von Datumsformaten beachten). Ein kurzer Blick genügt, um die verwendeten Zeichentrenner, Tausendertrennzeichen, Datumsformate, die Kodierung fehlender Werte und die Unicode-Kodierung (wie UTF-8) zu identifizieren.

 Tip ??

**Benutzen Sie die Dokumentation!** Auf diese Weise erhalten Sie einen vollständigen Überblick über standardmäßig gesetzte und optional verfügbare Parameter. Außerdem erkennen Sie Änderungen in der Programmausführung und vermeiden so unerwartete Fehler.

  
● *Added in version 1.5.0:* Support for `defaultdict` was added.  
Specify a `defaultdict` as input where the default determines the  
`dtype` of the columns which are not explicitly listed.

(a) Neuerung in Python

  
● *Deprecated since version 2.0.0:* A strict version of this argument  
is now the default, passing it has no effect.

(a) Abkündigung in Python

# 51 Grundlagen: Merkmale von Datensätzen

! Important ??: Datensatz

Ein Datensatz ist eine Sammlung zusammengehöriger Daten. Datensätze enthalten einer oder mehreren Variablen zugeordnete Werte. Jeder Datensatz besitzt ein technisches Format, eine Struktur, mindestens eine Variable und mindestens einen Wert.

## 51.1 Technisches Format

- 
- 
- 
-

## 51.2 Struktur

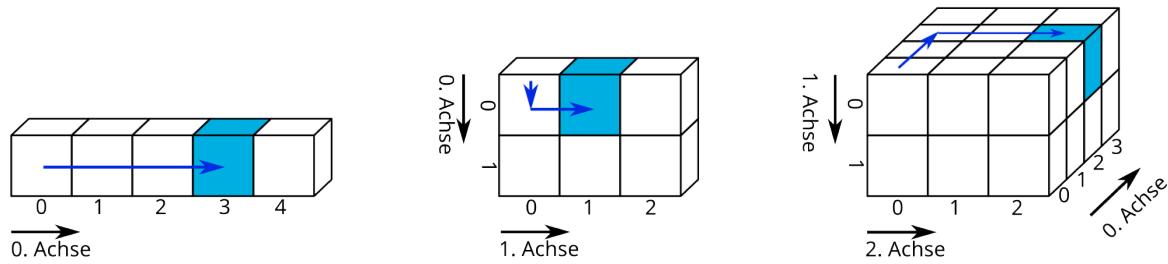


Figure 51.1: n-dimensionale Datensätze

[CC-BY-4.0](#)

[GitHub](#)

### 51.2.1 Eindimensionale Datensätze

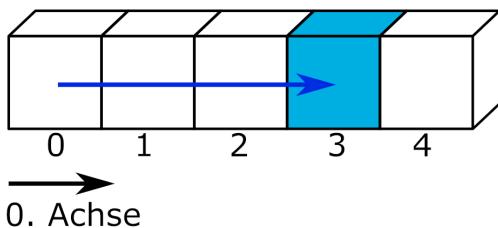


Figure 51.2: eindimensionale Datensätze

[CC-BY-4.0](#)

[GitHub](#)

```
print( *( Augen := [6, 2, 1, 2] ) )
print(f"Das Würfelergebnis an Indexposition 2 lautet: {Augen[2]}")
```

### 51.2.2 Eindimensionale Daten einlesen mit Python

Werkzeugbaustein Python



Tip ???: Kleine Wiederholung: Funktionen und Methoden der Pythonbasis

- Die Funktion `os.getcwd()` aus dem Modul os gibt das aktuelle Arbeitsverzeichnis aus, mit der Funktion `os.getcwd(pfad)` kann es gewechselt werden.
- Die Funktion `open(dateipfad, mode = 'r')` öffnet eine Datei im Lesemodus und gibt ein Dateiobjekt zurück.
- Informationen zum Dateiobjekt können durch Ausgabe verschiedener Attribute abgerufen werden: `dateiobjekt.name`, `os.path.basename(dateiobjekt.name)`, `dateiobjekt.closed`, `dateiobjekt.mode`, `dateiobjekt.encoding`
- Das Dateiobjekt kann mit Methoden wie `dateiobjekt.read()`, `dateiobjekt.readline()`, `dateiobjekt.readlines()` oder der Funktion `list(dateiobjekt)` ausgelesen werden.
- Die Methode `dateiobjekt.close()` schließt die Datei und gibt sie somit wieder für andere Programme frei.

- 
- 
-

💡 Tip ??: Musterlösung python.txt

```
dateipfad = "01-daten/" + "python.txt"
dateiobjekt = open(dateipfad, mode = 'r')

# Enkodierung der Datei bestimmen
print(f"Die Enkodierung der Datei lautet: {dateiobjekt.encoding}")

# Text ausgeben
text_als_liste = list(dateiobjekt)

for i in range(1, len(text_als_liste)):
    print(text_als_liste[i])

# Datei schließen.
dateiobjekt.close()
```

Die Enkodierung der Datei lautet: UTF-8

Python ist eine universell nutzbare, üblicherweise interpretierte, höhere Programmiersprache.

Python wurde mit dem Ziel größter Einfachheit und Übersichtlichkeit entworfen. Dies wird von

Van Rossum legte bei der Entwicklung großen Wert auf eine Standardbibliothek, die überschaubar

Auszug aus [https://de.wikipedia.org/wiki/Python\\_\(Programmiersprache\)](https://de.wikipedia.org/wiki/Python_(Programmiersprache)), abgerufen am 20.02.2023

Enkodierung UTF-8 auswählen.

```
# Mit europäischen Sonderzeichen kompatible Enkodierung UTF-8 wählen
dateiobjekt = open(dateipfad, mode = 'r', encoding = 'utf-8')

# Text ausgeben
text_als_liste = list(dateiobjekt)

for i in range(1, len(text_als_liste)):
    print(text_als_liste[i])

# Datei schließen.
dateiobjekt.close()
```

Python ist eine universell nutzbare, üblicherweise interpretierte, höhere Programmiersprache.

Python wurde mit dem Ziel größter Einfachheit und Übersichtlichkeit entworfen. Dies wird von

Van Rossum legte bei der Entwicklung großen Wert auf eine Standardbibliothek, die überschaubar

Auszug aus [https://de.wikipedia.org/wiki/Python\\_\(Programmiersprache\)](https://de.wikipedia.org/wiki/Python_(Programmiersprache)), abgerufen am 20.02.2018

### 51.2.3 Zweidimensionale Datensätze

Pandas

Werkzeugbaustein

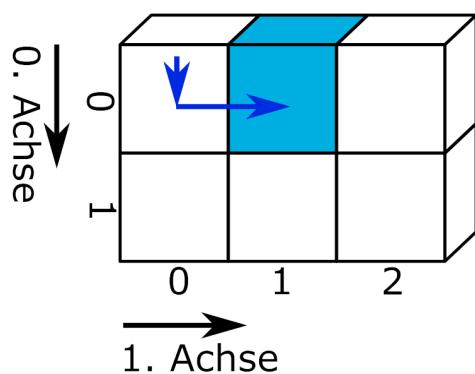


Figure 51.3: zweidimensionaler Datensatz

CC-BY-4.0

[GitHub](#)

```
import pandas as pd

messung1 = pd.DataFrame({'Name': ['Hans', 'Elke', 'Jean', 'Maya'], 'Geburtstag': ['26.02.',

messung1
```

```
print(f"Jean würfelte {messung1.iloc[2, 3]} Augen")
```

```
print(f"Jean würfelte {messung1['Summe Augen'][2]} Augen")
```

### ⚠ Warning ??: Verkettete Indexierung

Die verkettete Indexierung erzeugt in Pandas abhängig vom Kontext eine Kopie des Objekts oder greift auf den Speicherbereich des Objekts zu. Mit Pandas 3.0 wird die verkettete Indexierung nicht mehr unterstützt, das Anlegen einer Kopie wird zum Standard werden. Weitere Informationen erhalten Sie im zitierten Link.

“Whether a copy or a reference is returned for a setting operation, may depend on the context. This is sometimes called `chained assignment` and should be avoided. See [Returning a View versus Copy](#).”

([Pandas Dokumentation](#))

#### 51.2.4 long- und wide-Format

```
messung1 = pd.DataFrame({'Name': ['Hans', 'Elke', 'Jean', 'Maya', 'Hans'], 'Geburtstag': ['2019-01-01', '2019-01-01', '2019-01-01', '2019-01-01', '2019-01-01']})
```

```
messung1
```

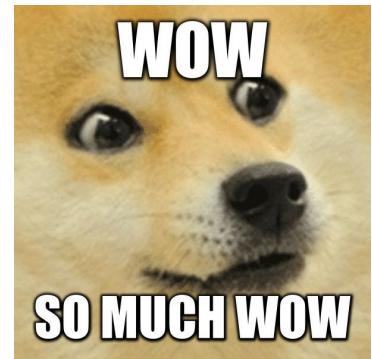
```
messung1_long = pd.melt(messung1, id_vars = ['Name', 'Geburtstag'])
```

```
messung1_long
```

```
messung1_all_id = pd.melt(messung1, id_vars = ['Name', 'Geburtstag', 'Würfelfarbe'])

messung1_all_id
```

	Name	Geburtstag	Würfelfarbe	variable	value
0	Hans	26.02.	rosa	Summe Augen	12
1	Elke	14.03.	rosa	Summe Augen	17
2	Jean	30.12.	blau	Summe Augen	8
3	Maya	07.09.	gelb	Summe Augen	23
4	Hans	11.11.	rosa	Summe Augen	7



💡 Tip ??: Antwort

Der Befehl `messung1_all_id = pd.melt(messung1, id_vars = ['Name', 'Geburtstag', 'Würfelfarbe', 'Summe Augen'])` produziert einen leeren Dataframe, weil keine gemessenen Werte verbleiben.

```
messung1_no_id = pd.melt(messung1)

messung1_no_id
```

```
# pd.pivot() benötigt einen Index oder benutzt den bestehenden Index, des melted_df, der zu #
# Deshalb eine zusätzliche Spalte in messung1_no_id einfügen
## einfach: messung1_no_id['new_index'] = list(range(0, 5)) * 4
## allgemein: messung1_no_id['new_index'] = messung1_no_id.groupby('variable').cumcount()

# Spalte new_index einfügen
messung1_no_id['new_index'] = messung1_no_id.groupby('variable').cumcount()
print (f"Der Datensatz im long-Format mit zusätzlicher Spalte new_index:\n{messung1_no_id}")

# casting
messung1_cast = pd.pivot(messung1_no_id, index = 'new_index', columns = 'variable', values =
print(f"\nDer Datensatz im wide-Format:\n{messung1_cast}")
```

```
# Spalten anordnen, Index zurücksetzen
messung1_cast = messung1_cast[['Name', 'Geburtstag', 'Würzelfarbe', 'Summe Augen']]
messung1_cast.reset_index(drop = True, inplace = True)
messung1_cast.rename_axis(None, axis = 1, inplace = True)

print(f"\nDer Datensatz im wide-Format mit zurückgesetztem Index:\n\n{messung1_cast}")
```

💡 Tip ??: identifizierende und gemessene Variablen

Auch wenn Sie mit Datensätzen im wide-Format arbeiten, ist die Unterscheidung identifizierender und gemessener Variablen nützlich, um Datensätze zu organisieren. siehe ??

### 51.2.5 Übung zweidimensionale Datensätze

💡 Tip ??: Musterlösung zweidimensionale Datensätze

```
# Spalte new_index einfügen
messung1_long['new_index'] = messung1_long.groupby('variable').cumcount()

# casting
messung1_long_cast = pd.pivot(messung1_long, index = 'new_index', columns = 'variable', val

# Spalten anordnen, Index zurücksetzen
messung1_long_cast = messung1_long[['Name', 'Geburtstag', 'Würfelfarbe', 'Summe Augen']]
messung1_long_cast.reset_index(drop = True, inplace = True)
messung1_long_cast.rename_axis(None, axis = 1, inplace = True)

messung1_long_cast
```

	Name	Geburtstag	Würzelfarbe	Summe Augen
0	Hans	26.02.	rosa	12
1	Elke	14.03.	rosa	17
2	Jean	30.12.	blau	8
3	Maya	07.09.	gelb	23
4	Hans	11.11.	rosa	7

### 51.2.6 Drei- und mehrdimensionale Datensätze

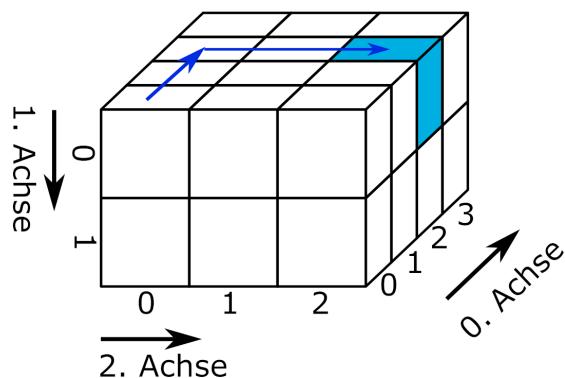


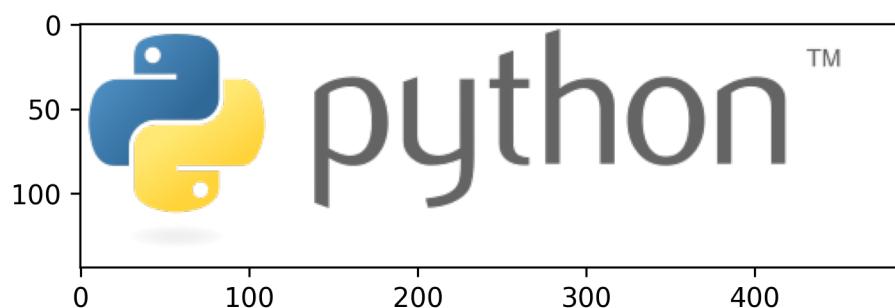
Figure 51.4: dreidimensionale Datensätze

[CC-BY-4.0](#)

[GitHub](#)

### 51.2.6.1 Bilddaten einlesen

```
import matplotlib.pyplot as plt  
  
logo = plt.imread(fname = '00-bilder/python-logo-and-wordmark-cc0-tm.png')  
  
plt.imshow(logo)
```



GPLv3  
<https://www.python.org/psf/trademarks/>  
wikimedia

```
print(type(logo), "\n")
print(logo.shape)
```

```
print(logo[50:52, 50:52, : ])
```

### 51.2.7 Übung dreidimensionale Datensätze

 Tip ??: Musterlösung dreidimensionale Datensätze

```
kanal = ["Rotkanal", "Grünkanal", "Blaukanal"]

plt.figure(figsize = (9, 6))

for i in range(3):

    plt.subplot(1, 4, i + 1)
    plt.imshow(logo[ :, :, i], cmap = 'Greys_r')
    plt.title(label = kanal[i])

plt.colorbar(shrink = 0.15)

plt.tight_layout()
plt.show()
```

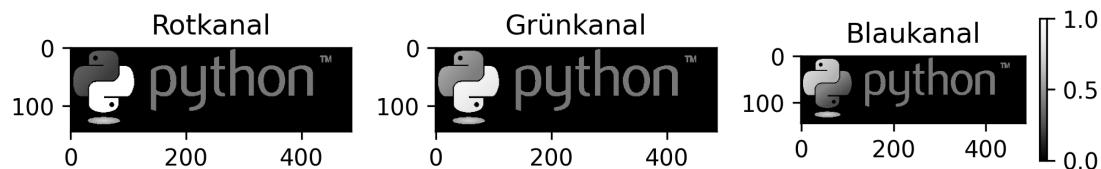


Figure 51.5: Farbkanäle des Pythonlogos

Möglicherweise wundern Sie sich, warum der Bildhintergrund in jedem Farbkanal schwarz ist. Die Ursache finden Sie im nächsten Tipp.

#### Tip 51.6: Erklärung Bildhintergrund

Der Bildhintergrund hat in allen Kanälen, auch im Alphakanal, den Farbwert 0. Dieser Teil des Bildes ist deshalb vollständig transparent und wird vom Hintergrund der Internetseite ausgefüllt. Der Bildhintergrund des Logos wirkt deshalb weiß.

```

# Alphakanal
plt.imshow(logo[ :, :, 3], cmap = 'Greys_r')
plt.title(label = 'Alphakanal')
plt.colorbar(shrink = 0.4)

plt.show()

# Die ersten zwei Zeilen und Spalten des Logos
print(logo[0:2, 0:2, : ])

```

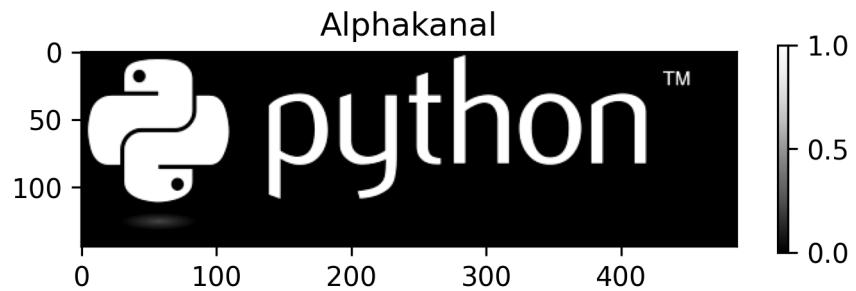


Figure 51.6: Alphakanal des Pythonlogos

```

[[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]]

```

### 51.3 Datentyp

[Dokumentation](#)

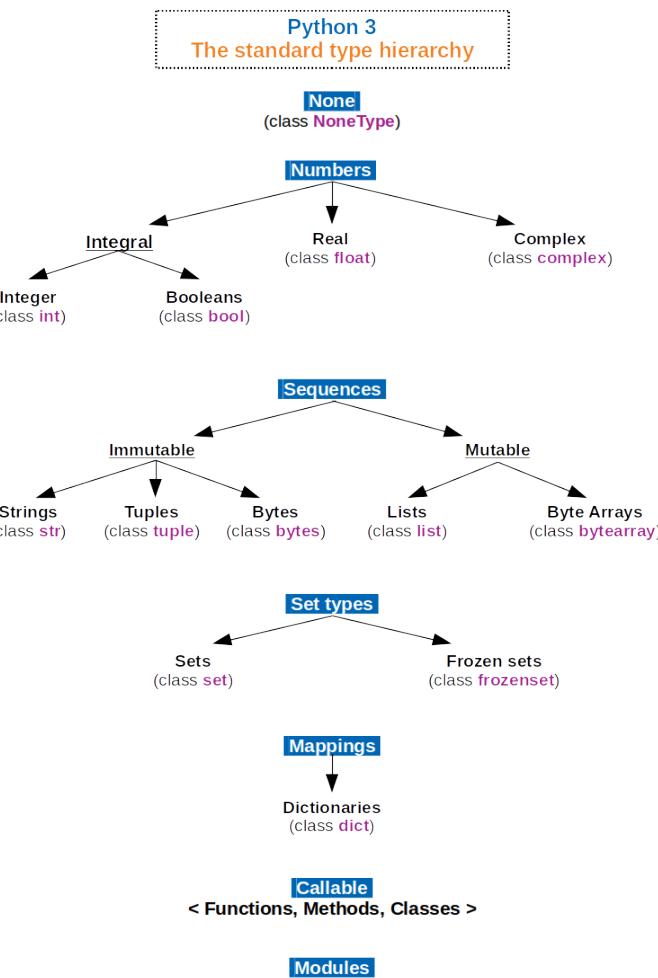


Figure 51.7: Datentypen in Python

wikimedia

CC BY SA 4.0

- 
- 
-

- 

- Pandas

**i** Note ??: Datentypabhängige Operationen und Funktionen

```
# Der Operator + bewirkt die Addition von Zahlen  
print(1 + 13)  
  
# Der Operator + bewirkt auch das Verketten von strings  
print(str(1) + str(13))
```

14

113

Die Sortierfunktion arbeitet abhängig vom Datentyp.

```
# Liste von Monatskürzeln erstellen  
dates = pd.Series(['07.06.2000', '12.01.2000', '11.02.2000', '04.09.2000', '10.03.2000',  
dates = pd.to_datetime(dates, format = '%d.%m.%Y');  
  
print(f"Eine unsortierte Liste von Monatskürzeln:\n{list(dates.dt.strftime('%b'))}")  
  
print(f"\nDie Liste alphabetisch sortiert:\n{sorted(list(dates.dt.strftime('%b')))}")  
  
print(f"\nDie Liste als datetime-Objekt sortiert:\n{list(dates.sort_values().dt.strftime('%b'))}")
```

Eine unsortierte Liste von Monatskürzeln:

['Jun', 'Jan', 'Feb', 'Sep', 'Mar', 'Oct', 'Apr', 'May', 'Jul', 'Aug', 'Nov', 'Dec']

Die Liste alphabetisch sortiert:

['Apr', 'Aug', 'Dec', 'Feb', 'Jan', 'Jul', 'Jun', 'Mar', 'May', 'Nov', 'Oct', 'Sep']

Die Liste als datetime-Objekt sortiert:

['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

 Tip ??: Datentyp kontrollieren und plausibilisieren

Beim Einlesen von Datensätzen ist es wichtig, die korrekte Erkennung der Datentypen zu kontrollieren bzw. aktiv zu steuern. Weitere Methoden für die formale Prüfung des Datentyps und für die Kontrolle des Wertebereichs werden in Chapter ?? vorgestellt.

### 51.3.1 Fehlende Werte

#### 51.3.1.1 Nullwert None

```
print(type(None))
```

```
leere_liste = []
leere_liste == None
```

```
# Operationen mit None führen zu Fehlermeldungen
try:
    print(None + 1)
except TypeError as error:
    print("Der übergebene Wert führt zu der Fehlermeldung:\n", error)
else:
    print(None + 1)
```

```
# Eine Ausnahme ist die Umwandlung in strings
a = None
print("\nprint(a) gibt den Nullwert zurück:\n", a, sep = "")

print("\nstr(a) gibt eine Zeichenkette zurück:")
str(a)
```

### 51.3.1.2 NaN

```
my_series = pd.Series([1, 2, 4, 8])
my_series.diff()
```

```
print(type(float('NaN')))
```

```
print(float('NaN') + 1)
```

```
# Python-Basis
print("sum():", sum([1, 2, float('NaN'), 4]), "\n")
print("max():", max([1, 2, float('NaN'), 4]), "\n")
print("any():", any([1, 2, float('NaN'), 4]), "\n")

# Pandas
daten_mit_nan = pd.Series([1, 2, float('NaN'), 4])
print(daten_mit_nan + 1)
print("\nSumme des Datensatzes:", daten_mit_nan.sum())
```

### ⚠ Warning ??: Achtung Logik!

Die logische Abfrage fehlender Werte unterscheidet sich für `None` und `NaN`.

```
bool_values = [None, float('NaN')]

for element in bool_values:
    bool_value = bool(element)
    print("Wahrheitswert von", element, "ist", bool_value)
```

Wahrheitswert von `None` ist `False`  
Wahrheitswert von `nan` ist `True`

Dies gilt auch für die Wertgleichheit.

```
for element in bool_values:
    result = element == element
    print("Wertgleichheit von", element, "ist", result)
```

Wertgleichheit von `None` ist `True`  
Wertgleichheit von `nan` ist `False`

### 51.3.2 Fehlende Werte in der Praxis

- 
- 
-

### Tip ??: fehlende Werte

Die Identifizierung und ggf. Bereinigung fehlender Werte ist ein wichtiger Schritt beim Einlesen strukturierter Datensätze. Dabei hilft es, die gängigen Kennzeichnungen für fehlende Werte zu kennen und sich über die Konventionen des jeweiligen Dateiformats bzw. der jeweiligen Disziplin zu informieren. Dennoch ist manchmal ein gewisser Spürsinn unerlässlich. Geeignete Funktionen zur Identifizierung fehlender Werte werden in Chapter ?? vorgestellt.

## 51.4 Metadaten

- 
- 
- 
- 
- 
- 
- 

[The HDF Group Help Desk](#)

### Tip ??: Metadaten

Insbesondere vor dem Einlesen komplexer Datensätze sollten Begleitmaterialien, sofern vorhanden, studiert werden.

## 51.5 Tidy data

“Tidy datasets are all alike, but every messy dataset is messy in its own way.” [@R-for-Data-Science, Kapitel 5 Data tidying]

### ! Important ??: tidy data

“Das System tidy data besteht aus drei Regeln:

1. Jede Variable ist eine Spalte; jede Spalte ist eine Variable.
2. Jede Beobachtung ist eine Zeile; jede Zeile ist eine Beobachtung.
3. Jeder Wert ist eine Zelle; jede Zelle ist ein einzelner Wert.”

[@R-for-Data-Science, Kapitel 5 Data tidying, eigene Übersetzung]

## 52 Die Module NumPy und Pandas

- NumPy

[siehe Dokumentation](#)

—

[siehe Dokumentation](#)

- Pandas

—

```
import numpy as np
import pandas as pd

# Deklarieren der Anzahl der Nachkommastellen
pd.set_option("display.precision", 2)
```

 Tip ??: Arbeiten mit NumPy und Pandas

Ob Sie mit NumPy oder mit Pandas arbeiten, hängt von dem vorliegenden Datensatz und persönlichen Präferenzen ab.

Das Paket Pandas erlaubt es, Daten aus verschiedenen Quellen wie CSV-Dateien oder Excel-Tabellen und mit unterschiedlichen Datentypen in einen DataFrame zu laden. Anschließend können diese mit wenigen Befehlen untersucht und umstrukturiert werden. Komplexe Operationen wie das Umformen von Datensätzen, das Gruppieren und Aggregieren von Daten sowie das Filtern und Sortieren sind effizient möglich. Bis auf wenige Ausnahmen sind Pandas und NumPy zueinander kompatibel. Es spricht nichts dagegen, Ihre Daten mit Pandas vorzubereiten und anschließend mit NumPy auszuwerten.

## 52.1 Datentypen

[Dokumentation NumPy](#)

[Dokumentation](#)

- Kategorie
- Zeitzonenbewusstes Datumsformat

Pandas

## 52.2 Dateien lesen und schreiben

## **52.3 NumPy**

- 
- 

## **52.4 Pandas**

[Pandas Dokumentation](#)

## 52.5 Datentypen erkennen und festlegen

### 52.5.1 NumPy

[NumPy Dokumentation](#)

```
dateipfad = '01-daten/TC01.csv'  
daten = np.loadtxt(dateipfad)
```

💡 Tip ??: Musterlösung Datentypumwandlung

```
# Ausgabe des Datentyps
print(daten.dtype)

# Umwandlung in Ganzzahl
daten_int = daten.astype('int')

# Prüfen auf Datenverlust
prüfsumme = daten - daten_int
print(f"Differenz daten - daten_int: {prüfsumme.sum()}")

float64
Differenz daten - daten_int: 664.0
```

## 52.5.2 Pandas

### 52.5.2.1 Zahnwachstum bei Meerschweinchen

<https://doi.org/10.1093/jn/33.5.491>

- 
- 

 Tip ??: Musterlösung Meerschweinchen

```
dateipfad = "01-daten/ToothGrowth.csv"
meerschweinchen = pd.read_csv(filepath_or_buffer = dateipfad, sep = ',', header = 0, \
    names = ['ID', 'len', 'supp', 'dose'], dtype = {'ID': 'int', 'len': 'float', 'dose': 'float'})

# Ausgabe jedes sechsten Werts
meerschweinchen.iloc[meerschweinchen.index % 6 == 0]
```

	ID	len	supp	dose
0	1	4.2	VC	0.5
6	7	11.2	VC	0.5
12	13	15.2	VC	1.0
18	19	18.8	VC	1.0
24	25	26.4	VC	2.0
30	31	15.2	OJ	0.5
36	37	8.2	OJ	0.5
42	43	23.6	OJ	1.0
48	49	14.5	OJ	1.0
54	55	24.8	OJ	2.0

```
print(meerschweinchen.dtypes)
```

```
ID      int64
len     float64
supp    category
dose    float64
dtype: object
```

### 52.5.2.2 Nützliche Funktionen für die deskriptive Datenanalyse

```
print(meerschweinchen.columns)
meerschweinchen.columns = ['ID', 'Länge', 'Verabreichung', 'Dosis']
print(meerschweinchen.columns)
```

```
print(meerschweinchen.describe(), "\n")
print(meerschweinchen.describe(include = 'all'), "\n")
print(meerschweinchen.describe(include = ['float']), "\n")
```

```
meerschweinchen.count(axis = 'rows') # der Standardwert von axis ist 'rows'
```

```
meerschweinchen.info()
```

```
meerschweinchen['Dosis'].unique()
```

💡 Tip ???: Nützliche Funktionen

Pandas bietet einige praktische Funktionen, um eine eingelesene Datei zu kontrollieren. Machen Sie sich die Verwendung von `pd.dtypes` oder `pd.DataFrame.info()` zur Ange-wohnheit.

### 52.5.3 Aufgabe Datentypen

`pd.read_excel`

[https://www.gov.uk/government/uploads/system/uploads/attachment\\_data/file/670121/table\\_531.xls](https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/670121/table_531.xls)

💡 Tip ???: Musterlösung 5.3.1 (excl. taxes)

Überspringen der führenden Zeilen mit dem Argument `header = 8`. Auswahl des Tabel- lenblatts mit `sheet_name = "5.3.1 (excl. taxes)"` und Kontrolle der erkannten Da-tentypen mit `taxes.dtypes`

```
dateipfad = '01-daten/table_531.xlsx'

taxes = pd.read_excel(io = dateipfad, sheet_name = "5.3.1 (excl. taxes)", \
    header = 8)

taxes.dtypes
```

Year	int64
Austria	float64

```
Belgium          float64
Denmark          float64
Finland          float64
France           float64
Germany          float64
Greece           float64
Ireland          float64
Italy             float64
Luxembourg       float64
Netherlands      float64
Portugal          float64
Spain             float64
Sweden            float64
United Kingdom   float64
Australia         float64
Canada            float64
Czech Republic   float64
Hungary           float64
Japan              float64
Korea              float64
New Zealand       float64
Norway             float64
Poland             float64
Slovakia          float64
Switzerland       float64
Republic of Türkiye    object
USA                float64
IEA median        float64
UK relative to IEA median% float64
UK relative to IEA rank    int64
UK relative to G7 rank     int64
dtype: object
```

Werte in Spalte ‘Republic of Türkiye’ mit pd.unique() ansehen.

```
taxes['Republic of Türkiye'].unique()
```

```
array(['..', 2.043608174999997, 3.324858443999993, 3.2947581129644483,
       3.5628243387317866, 3.998334312, 3.838962582401693,
       4.2138469457789975, 3.775503630575527, 3.2804905218375238,
       3.783413840344277, 4.139259596071514, 4.196890742949158,
       4.658509330911754, 5.552842625063031, 4.316920402166109,
```

```

4.1586205264300675, 4.765321921741988, 4.060617948410105,
3.9191433658651307, 4.223710389549368, 4.481407237836746,
4.629981488840797, 5.2657882806931235, 5.109009847145703,
4.585007793872617, 4.769921255774284, 4.419433670846949,
4.428906151745361, 6.171573537217762, 7.192920543071899,
7.962417550086158, 7.035941949054445, 7.622058781522502,
7.644892388451444, 6.47006818181818, 5.968380462724936,
6.379514692256784, 5.537541821623266, 5.248709303933227,
6.9100519994521274, 6.670900808798327, 5.864171132090749,
13.928251887312259, 11.123594768114717], dtype=object)

```

Zeichenkette ‘.’ entfernen und Datentyp mit Methode `pd.astype('float64')` ändern.

- Variante 1: als fehlenden Wert beim Einlesen deklarieren.
- Variante 2: Nach dem Einlesen Indexposition bestimmen und Wert ersetzen. Das verkettete Slicing `df["col"] [row_indexer] = value` wird mit der Pandas Version 3.0 nicht mehr unterstützt und gibt deshalb eine Fehlermeldung aus. Künftig ist folgende Syntax zu verwenden: `df.loc[row_indexer, "col"] = value`.

```

# Variante 1: '...' als fehlenden Wert deklarieren
# taxes = pd.read_excel(io = dateipfad, sheet_name = "5.3.1 (excl. taxes)", \
# header = 8, na_values = ['...'])

# Variante 2: Index des Werts bestimmen und mit np.nan überschreiben
indexposition = taxes['Republic of Türkiye'] == '...'

taxes.loc[indexposition, 'Republic of Türkiye'] = np.nan
taxes['Republic of Türkiye'] = taxes['Republic of Türkiye'].astype('float64')

taxes.dtypes

```

Year	int64
Austria	float64
Belgium	float64
Denmark	float64
Finland	float64
France	float64
Germany	float64
Greece	float64
Ireland	float64
Italy	float64

```
Luxembourg          float64
Netherlands         float64
Portugal            float64
Spain               float64
Sweden              float64
United Kingdom     float64
Australia           float64
Canada              float64
Czech Republic     float64
Hungary             float64
Japan               float64
Korea               float64
New Zealand         float64
Norway              float64
Poland              float64
Slovakia            float64
Switzerland          float64
Republic of Türkiye float64
USA                 float64
IEA median          float64
UK relative to IEA median% float64
UK relative to IEA rank    int64
UK relative to G7 rank      int64
dtype: object
```

## 52.6 Umgang mit fehlenden Werten

### 52.6.1 NumPy

```

dateipfad = '01-daten/TC01.csv'
daten_ohne_fehlende_werte = np.loadtxt(dateipfad)

print("Daten:", daten_ohne_fehlende_werte)
print("Struktur:", daten_ohne_fehlende_werte.shape, "dtype:", daten_ohne_fehlende_werte.dtype)

```

```

dateipfad = '01-daten/TC01_double_hyphen.csv'

try:
    daten_double_hyphen = np.loadtxt(dateipfad)
except ValueError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print("Daten mit fehlenden Werten '--':", daten_double_hyphen, "dtype:", daten_double_hyphen)

```

### 52.6.1.1 Die Funktion `np.genfromtxt()`

```
dateipfad = '01-daten/TC01_double_hyphen.csv'
daten_double_hyphen = np.genfromtxt(dateipfad, missing_values = '--', filling_values = np.nan)

print("\nDaten mit fehlenden Werten '--':", daten_double_hyphen)
print("Struktur:", daten_double_hyphen.shape, "dtype:", daten_double_hyphen.dtype)
```

```
daten_differenz = daten_ohne_fehlende_werte - daten_double_hyphen
print(daten_differenz)
```

### **i** Note ??: Leere Zellen mit np.genfromtxt()

Enthält eine Datei leere Zellen, können diese nicht eingelesen werden, da diese automatisch übersprungen werden.

```
# Datei ohne Markierung fehlender Werte
dateipfad = '01-daten/TC01_empty_lines.csv'
daten_empty_lines = np.genfromtxt(dateipfad, missing_values = '', filling_values = np.nan)

print("\nDaten mit fehlenden Werten '':", daten_empty_lines)
print("Struktur:", daten_empty_lines.shape, "dtype:", daten_empty_lines.dtype)
```

```
Daten mit fehlenden Werten '': [20.1 20.1 20.1 ... 24.3 24.2 24.2]
Struktur: (1511,) dtype: float64
```

Das Array ist zwei Elemente kürzer. Die Subtraktion von einem längeren NumPy-Array scheitert mit einer Fehlermeldung.

```

try:
    result = daten_ohne_fehlende_werte - daten_empty_lines
except ValueError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(result)

```

Die Eingabe führt zu der Fehlermeldung:

```
operands could not be broadcast together with shapes (1513,) (1511,)
```

In diesem Fall muss auf die Stringbearbeitung aus der Python-Basis zurückgegriffen werden. Die bearbeitete Liste kann wie gewohnt mit `np.genfromtxt()` eingelesen werden.

```

# Einlesen über Datenobjekt
datenobjekt_empty_lines = open(dateipfad, 'r', encoding = 'utf-8')
daten_empty_lines = datenobjekt_empty_lines.readlines()
datenobjekt_empty_lines.close()

print("Das ausgelesene Datenobjekt (Ausschnitt):\n", daten_empty_lines[0:10])

# Stringbearbeitung mit replace('\n', '')
for i in range(len(daten_empty_lines)):

    if daten_empty_lines[i] == '\n':
        daten_empty_lines[i] = 'platzhalter'
    else:
        daten_empty_lines[i] = daten_empty_lines[i].replace('\n', '')

print("\nNach der Stringbearbeitung (Ausschnitt):\n", daten_empty_lines[0:10])

# Einlesen mit np.genfromtxt
daten_empty_lines = np.genfromtxt(daten_empty_lines, missing_values = 'platzhalter', filling_value = 'platzhalter')
print("\nDaten mit fehlenden Werten '':", daten_empty_lines)
print("Struktur:", daten_empty_lines.shape, "dtype:", daten_empty_lines.dtype)

Das ausgelesene Datenobjekt (Ausschnitt):
['# Temperatur in C\n', '20.1\n', '\n', '20.1\n', '20.1\n', '20.1\n', '\n', '20.1\n', '20.1\n', '20.1\n', '20.1\n']

Nach der Stringbearbeitung (Ausschnitt):
['# Temperatur in C', '20.1', 'platzhalter', '20.1', '20.1', '20.1', 'platzhalter', '20.1', '20.1', '20.1', '20.1']

Daten mit fehlenden Werten '': [20.1  nan 20.1 ... 24.3 24.2 24.2]

```

```
Struktur: (1513,) dtype: float64
```

## NumPy Dokumentation

**i** Note ??: Leere Zellen in mehreren Spalten mit np.genfromtxt()

Ohne Spezifikation des Arguments `delimiter` wird nur eine Spalte eingelesen, die ausschließlich `np.nan` enthält.

```
# ohne Spezifikation von delimiter
dateipfad = '01-daten/TC01_missing_values_multi_column.csv'
daten_empty_lines2 = np.genfromtxt(dateipfad, missing_values = '', filling_values = np.nan)

print("Struktur:", daten_empty_lines2.shape, "dtype:", daten_empty_lines2.dtype)
print("Die ersten drei Zeilen:\n", daten_empty_lines2[0:3])
```

Struktur: (1513, 1) dtype: float64

Die ersten drei Zeilen:

```
[[nan]
[nan]
[nan]]
```

Wird das Argument `delimiter = ','` übergeben, wird die Datei korrekt eingelesen.

```
# mit Spezifikation von delimiter
daten_empty_lines2 = np.genfromtxt(dateipfad, delimiter = ',', missing_values = '', filling_values = np.nan)

print("Struktur:", daten_empty_lines2.shape, "dtype:", daten_empty_lines2.dtype)
print("\nDaten mit fehlenden Werten '':\n", daten_empty_lines2)
```

Struktur: (1513, 2) dtype: float64

Daten mit fehlenden Werten '':

```
[[20.1 20.1]
[ nan  nan]
[20.1 20.1]
...
[24.3 24.3]]
```

```
[24.2 24.2]  
[24.2 24.2]]
```

### 52.6.1.2 Fehlende Werte in NumPy erzeugen, prüfen, finden, ersetzen, löschen

- 
- 
- 

#### **i** Note ??: Die Funktion np.argwhere()

Eine andere Funktion, um die Indexposition eines Werts zu bestimmen, ist die Funktion `np.argwhere()`. Der Aufruf der Funktion `np.argwhere(np.isnan(array))` gibt ein NumPy-Array mit den Indexposition Elementen mit dem Wert `nan` zurück.

```
array = np.array([[1, np.nan, np.nan], [4, 5, np.nan]])  
print(array)  
  
np.argwhere(np.isnan(array))
```

```
[[ 1. nan nan]  
 [ 4. 5. nan]]  
  
array([[0, 1],  
       [0, 2],  
       [1, 2]])
```

Das mit `np.argwhere()` erzeugte Array ist aber nicht geeignet, um Arraybereiche auszuwählen.

```

try:
    array[np.argwhere(np.isnan(array))]
except IndexError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(array[np.argwhere(np.isnan(array))])

```

Die Eingabe führt zu der Fehlermeldung:  
index 2 is out of bounds for axis 0 with size 2

Zum Vergleich mit `np.nonzero()`

```

try:
    array[np.nonzero(np.isnan(array))]
except IndexError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(array[np.nonzero(np.isnan(array))])

```

[nan nan nan]

### Warning 52.1: Die Funktion `np.argwhere()`

Die Auswahl von Array-Bereichen mit `np.argwhere()` funktioniert für eindimensionale Arrays.

```

array = np.array([1, np.nan, np.nan, 4, 5])

try:
    array[np.argwhere(np.isnan(array))]
except IndexError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(array[np.argwhere(np.isnan(array))])

```

[[nan]  
[nan]]

-

### **i** Note ??: Wertzuweisung mit logischem Vektor

Die Ersetzung eines bestimmten Werts ist auch durch die Auswahl bestimmter Array-Bereiche durch einen logischen Vektor möglich.

```
a = np.array([1, 2, 3, np.nan, 5, 6, np.nan])

b = np.isnan(a)

print(b)

a[b] = 0

print(a)

[False False False  True False False  True]
[1. 2. 3. 0. 5. 6. 0.]
```

Dabei können mehrere Bedingungen mit der Funktion `np.logical_or(x1, x2)` als logisches ODER kombiniert werden.

```
a = np.array([1, 2, 3, np.nan, 5, 6, np.nan])

bedingung1 = np.isnan(a)

bedingung2 = a >= 5

bedingung = np.logical_or(bedingung1, bedingung2)

a[bedingung] = 0

print(a)

[1. 2. 3. 0. 0. 0. 0.]
```

Auch ein logisches UND ist möglich (aber in Verbindung mit `np.nan` nicht sinnvoll). Der Operator `*` bewirkt das gleiche wie der logische Operator `and` oder die Funktion `np.logical_and(x1, x2)`.

```
a = np.array([1, 2, 3, np.nan, 5, 6, np.nan])  
  
bedingung1 = a < 4  
  
bedingung2 = a >= 1  
  
bedingung = bedingung1 * bedingung2  
  
a[bedingung] = 0  
  
print(a)  
  
[ 0.  0.  0. nan  5.  6. nan]
```

•

```
np_array_with_none = np.array([1, 2, None, 4])  
print(np_array_with_none, np_array_with_none.dtype)
```

### 💡 Tip ??: Lösung

Eine logische Abfrage von `None` ist möglich. Auf diese Weise kann ein logisches Array erzeugt werden, das zur Auswahl der Indexpositionen verwendet wird, deren Werte ersetzt werden sollen.

```

np_array_with_none = np.array([1, 2, None, 4])
print(np_array_with_none)

np_array_with_nan = np_array_with_none.copy()

print(f"\nArray mit logischer Abfrage von None:\n{np_array_with_none == None}")
np_array_with_nan[np_array_with_none == None] = np.nan
print(f"\nArray mit None ersetzt durch nan:\n{np_array_with_nan, np_array_with_nan.dtype}")

[1 2 None 4]

Array mit logischer Abfrage von None:
[False False  True False]

Array mit None ersetzt durch nan:
(array([1, 2, nan, 4], dtype=object), dtype('O'))

```

### 52.6.1.3 Operationen mit fehlenden Werten

[Dokumentation](#)

```

print(f"Array mit nan:\n{np_array_with_nan}\n")

print(f"Summe des Arrays:\n{np.sum(np_array_with_nan)}\n")

print(f"nan-Summe des Arrays:\n{np.nansum(np_array_with_nan)}\n")

print(f"kumulierte Summe des Arrays:\n{np.cumsum(np_array_with_nan)}\n")

print(f"kumulierte nan-Summe des Arrays:\n{np.nancumsum(np_array_with_nan)}\n")

```

## 52.6.2 Pandas

Pandas Dokumentation

```
dateipfad = '01-daten/TC01_double_hyphen.csv'

try:
    daten_double_hyphen = pd.read_csv(dateipfad, na_values = ['--'])
except ValueError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print("Daten mit fehlenden Werten '--':\n", daten_double_hyphen, daten_double_hyphen.shape)
```

```
dateipfad = '01-daten/TC01_empty_lines.csv'

try:
    daten_empty_lines = pd.read_csv(dateipfad, skip_blank_lines = False)
except ValueError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print("Daten mit fehlenden Werten '':\n", daten_empty_lines, daten_empty_lines.shape)
```

```
•  
•  
  
dateipfad = '01-daten/TC01_empty_lines.csv'  
  
try:  
    daten_empty_lines = pd.read_csv(dateipfad, skip_blank_lines = False, dtype = 'string')  
except ValueError as error:  
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)  
else:  
    print("Daten mit fehlenden Werten '':\n", daten_empty_lines, daten_empty_lines.shape)
```

**i** Note ???: pd.Series mit np.nan und pd.NA

Eine pd.Series mit np.nan wird automatisch in dtype: float64 umgewandelt:

```

try:
    test = pd.Series([1, 2, np.nan])
except TypeError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(test)

```

```

0    1.0
1    2.0
2    NaN
dtype: float64

```

Eine pd.Series mit pd.NA wird als dtype: object eingelesen:

```

try:
    test = pd.Series([1, 2, pd.NA])
except TypeError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(test)

```

```

0      1
1      2
2    <NA>
dtype: object

```

Der dtype kann für eine Series mit pd.NA festgelegt werden:

```

try:
    test = pd.Series([1, 2, pd.NA], dtype = 'Int32')
except TypeError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(test)

```

```

0      1
1      2
2    <NA>
dtype: Int32

```

Abhängig vom Datentyp kommt es auf den korrekten dtype (NumPy oder Pandas) an, erkennbar an der Groß- und Kleinschreibung. pd.NA mit Numpy-Fließkommazahl:

```

try:
    test = pd.Series([1, 2, pd.NA], dtype = 'float64')
except TypeError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(test)

```

Die Eingabe führt zu der Fehlermeldung:

float() argument must be a string or a real number, not 'NAType'

pd.NA mit Pandas-Fließkommazahl:

```

try:
    test = pd.Series([1, 2, pd.NA], dtype = 'Float64')
except TypeError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(test)

```

```

0      1.0
1      2.0
2    <NA>
dtype: Float64

```

np.nan mit Numpy-Fließkommazahl:

```

try:
    test = pd.Series([1, 2, np.nan], dtype = 'float64')
except TypeError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    print(test)

```

```

0      1.0
1      2.0
2      NaN
dtype: float64

```

np.nan mit Pandas-Fließkommazahl:

```
try:  
    test = pd.Series([1, 2, np.nan], dtype = 'Float64')  
except TypeError as error:  
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)  
else:  
    print(test)  
  
0      1.0  
1      2.0  
2    <NA>  
dtype: Float64
```

## ⚠ Warning ??: Achtung Logik!

Die logische Abfrage fehlender Werte unterscheidet sich für `None`, `np.nan` und `pd.NA`.

```
bool_values = [None, float('nan'), pd.NA]

for element in bool_values:
    try:
        bool_value = bool(element)
    except TypeError as error:
        print(error)
    else:
        print("Wahrheitswert von", element, "ist", bool_value)
```

```
Wahrheitswert von None ist False
Wahrheitswert von nan ist True
boolean value of NA is ambiguous
```

Dies gilt auch für die Wertgleichheit.

```
bool_values = [None, float('nan'), pd.NA]

for element in bool_values:
    try:
        result = element == element
    except TypeError as error:
        print(error)
    else:
        print("Wertgleichheit von", element, "ist", result)
```

```
Wertgleichheit von None ist True
Wertgleichheit von nan ist False
Wertgleichheit von <NA> ist <NA>
```

[Pandas Dokumentation](#)

### 52.6.2.1 Fehlende Werte in Pandas erzeugen, prüfen, finden, ersetzen, löschen

• Pandas

• Dokumentation

### 52.6.2.2 Operationen mit fehlenden Werten

```
print(pd.NA ** 0)
print(1 ** pd.NA)
```

```
print(pd.Series([pd.NA]).sum())
print(pd.Series([pd.NA]).prod())
```

```
print(pd.Series([pd.NA]).min())
print(pd.Series([pd.NA]).mean())
print(pd.Series([pd.NA]).cumsum())
print(pd.Series([pd.NA]).cumprod())
```

### 52.6.3 Aufgabe fehlende Werte

[https://opendata.dwd.de/  
climate\\_environment/CDC/observations\\_germany/climate/hourly/solar/stundenwerte\\_  
ST\\_01303\\_row.zip](https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/hourly/solar/stundenwerte_ST_01303_row.zip)

💡 Tip ??: Musterlösung fehlende Werte

Mit der Methode `df.info()` ist erkennbar, dass der Datensatz vollständig ist.

```
dateipfad = "01-daten/produkt_st_stunde_20230831_20240630_01303.txt"
solar = pd.read_csv(dateipfad, sep = ";")

solar.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7296 entries, 0 to 7295
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   STATIONS_ID    7296 non-null   int64  
 1   QN_592        7296 non-null   int64  
 2   ATMO_LBERG    7296 non-null   float64 
 3   FD_LBERG      7296 non-null   float64 
 4   FG_LBERG      7296 non-null   float64 
 5   SD_LBERG      7296 non-null   int64  
 6   ZENIT         7296 non-null   float64 
dtypes: float64(4), int64(3)
memory usage: 399.1 KB
```

Mit der Methode `df.describe()` wird die deskriptive Statistik für numerische Spalten erstellt.

```
solar.describe()
```

	STATIONS_ID	QN_592	ATMO_LBERG	FD_LBERG	FG_LBERG	SD_LBERG	ZENIT
count	7296.0	7296.0	7296.00	7296.00	7296.00	7296.00	7296.00
mean	1303.0	1.0	111.80	-31.23	-16.46	9.00	92.65
std	0.0	0.0	89.42	222.34	229.74	18.66	30.02
min	1303.0	1.0	-999.00	-999.00	-999.00	0.00	28.56
25%	1303.0	1.0	112.00	0.00	0.00	0.00	70.61
50%	1303.0	1.0	121.00	0.00	0.00	0.00	92.40
75%	1303.0	1.0	127.25	25.00	33.00	3.00	115.9
max	1303.0	1.0	150.00	182.00	348.00	60.00	151.4

Drei Spalten weisen als minimalen Wert -999 auf, der inhaltlich nicht sinnvoll ist. Wie oft kommt der Wert -999 in den Spalten vor?

```
counting_df = solar[['ATMO_LBERG', 'FD_LBERG', 'FG_LBERG']] == -999
print(counting_df.sum())
print("Summe:\t\t ", counting_df.sum().sum())
```

```
ATMO_LBERG      46
FD_LBERG       359
FG_LBERG       353
dtype: int64
Summe:        758
```

# 53 Zeitreihen

## 53.1 Datum und Zeit in NumPy und Pandas

## 53.2 NumPy

- ISO 8601

```
print(np.datetime64('2024'), np.datetime64('2024').dtype)
print(np.datetime64('2024-10-31'), np.datetime64('2024-10-31').dtype)
print(np.datetime64('2024-10-31T12:24:59.999'), np.datetime64('2024-10-31T12:24:59.999').dtype)
```

- 

[NumPy Dokumentation](#)

```
print(np.datetime64(10 * 1000, 'D'), np.datetime64(10 * 1000, 'D').dtype)
print(np.datetime64(1000 * 1000, 'h'), np.datetime64(1000 * 1000, 'h').dtype)
print(np.datetime64(1000 * 1000 * 1000, 's'), np.datetime64(1000 * 1000 * 1000, 's').dtype)
```

```
my_array = np.array(['2007-07-13', '2006-01-13', '2010-08-13'], dtype = 'datetime64[s]')
print(my_array, my_array.dtype)
```

```
np.arange('2005-02', '2005-03', dtype = 'datetime64[D]')
```

[NumPy Dokumentation](#)

### 53.3 Pandas

- 
- 

```
print(pd.to_datetime(1000, unit = 'D'))
print(pd.to_datetime(1000 * 1000, unit = 'h'))
print(pd.to_datetime(1000 * 1000 * 1000, unit = 's'))
```

- 

### ISO 8601

```
print(pd.to_datetime('2017'))
print(pd.to_datetime('2017-01-01T00'))
print(pd.to_datetime('2017-01-01 00:00:00'))
```

- 

### Doku- mentation strftime zur string-Formatierung

```
print(pd.to_datetime('Monday, 12. August `24', format = "%A, %d. %B ``y"))
print(pd.to_datetime('Monday, 12. August 2024, 12:15 Uhr CET', format = "%A, %d. %B %Y, %H:%M"))
```

- 

```
print(pd.to_datetime({'year':[2020, 2024], 'month': [1, 11], 'day': [1, 21]}), "\n")
print(pd.to_datetime(pd.DataFrame({'year':[2020, 2024], 'month': [1, 11], 'day': [1, 21]})))
```

- 
- 
- 
- Liste verfügbarer strings
- 

```
print(pd.date_range(start = '2017', end = '2024', periods = 3), "\n")  
print(pd.date_range(start = '2017', end = '2024', freq = 'YE'), "\n")  
print(pd.date_range(end = '2024', freq = 'h', periods = 3))
```

 Zeitdifferenzen in NumPy und Pandas

## 53.4 NumPy

Zeitdifferenzen werden mit dem Datentyp `timedelta64` abgebildet. Dieser wird wie `datetime64` durch Angabe einer Ganzzahl und einer Zeiteinheit angelegt.

```
np.timedelta64(1, 'D')
```

```
np.timedelta64(1, 'D')
```

Objekte der Typen `datetime64` und `timedelta64` ermöglichen es, Operationen mit Datum und Zeit durchzuführen (weitere Beispiele in der [NumPy-Dokumentation](#)).

```
print(np.datetime64('today') - np.datetime64('2000-01-01', 'D'))
print(np.datetime64('now') - np.datetime64('2000-01-01', 'h'))
```

```
print("\n\nEine einfache Zeitverschiebung:", np.datetime64('now') - np.timedelta64(1, 'h'))
print("Wie viele Tage hat die Woche?", np.timedelta64(1,'W') / np.timedelta64(1,'D'))
```

```
9552 days
825326748 seconds
```

```
Eine einfache Zeitverschiebung: 2026-02-25T08:25:48
Wie viele Tage hat die Woche? 7.0
```

## 53.5 Pandas

Zeitdifferenzen können zum einen wie in NumPy durch Angabe einer Ganzzahl und einer Zeiteinheit angelegt werden. Außerdem ist die Übergabe mit Argumenten möglich (zulässige Argumente sind: weeks, days, hours, minutes, seconds, milliseconds, microseconds, nanoseconds).

```
pd.Timedelta(1, 'D')
pd.Timedelta(days = 1, hours = 1)
```

```
Timedelta('1 days 01:00:00')
```

**Wichtig:** Anders als in NumPy werden Zeitdifferenzen in Monaten und Jahren nicht mehr von Pandas unterstützt.

```
try:  
    print( pd.Timedelta(1, 'YE'))  
except ValueError as error:  
    print(error)  
else:  
    print( pd.Timedelta(1, 'YE'))
```

```
invalid unit abbreviation: YE
```

Zum anderen können Zeitdifferenzen mit einer Zeichenkette erzeugt werden.

```
print(pd.Timedelta('10sec'))  
print(pd.Timedelta('10min'))  
print(pd.Timedelta('10hours'))  
print(pd.Timedelta('10days'))  
print(pd.Timedelta('10w'))
```

```
0 days 00:00:10  
0 days 00:10:00  
0 days 10:00:00  
10 days 00:00:00  
70 days 00:00:00
```

Mit Hilfe einer Zeitdifferenz können Zeitreihen leicht verschoben werden.

```
pd.date_range(start = '2024-01-01T00:00', end = '2024-01-01T02:00', freq = '15min') + pd.T
```

```
DatetimeIndex(['2024-01-01 00:30:00', '2024-01-01 00:45:00',  
                '2024-01-01 01:00:00', '2024-01-01 01:15:00',  
                '2024-01-01 01:30:00', '2024-01-01 01:45:00',  
                '2024-01-01 02:00:00', '2024-01-01 02:15:00',  
                '2024-01-01 02:30:00'],  
               dtype='datetime64[ns]', freq='15min')
```

Wie alt sind Sie in Tagen? Wie alt in Sekunden? Rechnen Sie mit NumPy oder Pandas.

### Tip 53.2: Tipp Pandas und Musterlösung Alter

Für eine elegante Lösung in Pandas schauen Sie sich die verfügbaren Methoden und Attribute von Timedelta-Objekten an.

```
dir(pd.Timedelta(0))
```

#### Tip 53.1: Musterlösung

Ersetzen sie in der Lösung die Zeichenkette 'YYYY-MM-DD' bzw., wenn Sie die Uhrzeit Ihrer Geburt kennen, die Zeichenkette 'YYYY-MM-DDTHH:MM' durch ihren Geburtstag.

##### 53.5.0.1 NumPy

In NumPy können die Schlüsselwörter `np.datetime64('today')` und `np.datetime64('now')` verwendet werden. Die Ausgabe ist in Tagen bzw. in Sekunden aufgelöst.

```
print(np.datetime64('today') - np.datetime64('YYYY-MM-DD', 'D'))  
print(np.datetime64('now') - np.datetime64('YYYY-MM-DDTHH:MM', 's'))
```

##### 53.5.0.2 Pandas

In Pandas werden die Schlüsselwörter `pd.to_datetime('today')` und `pd.to_datetime('now')` in Nanosekunden aufgelöst.

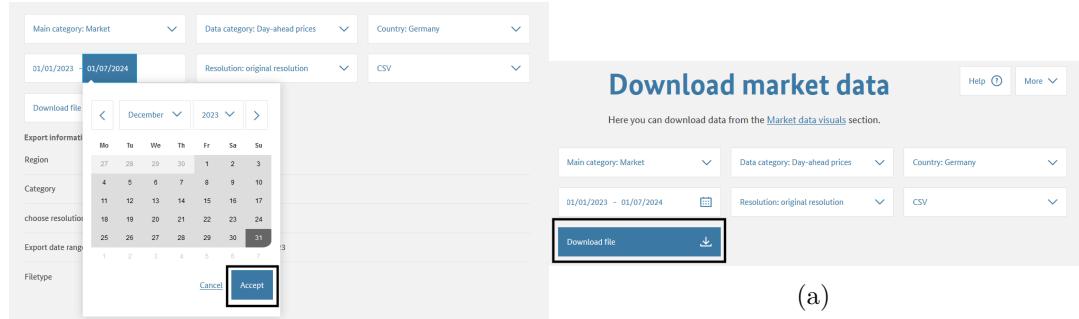
```
(pd.to_datetime('today') - pd.to_datetime('YYYY-MM-DD')).days  
(pd.to_datetime('now') - pd.to_datetime('YYYY-MM-DDTHH:MM')).total_seconds()
```

## 53.6 Zeitreihen einlesen

<https://www.smard.de/>

### **⚠ Warning ??: SMARD Daten herunterladen**

Beim der Auswahl des Zeitraums auf Daten in Originalauflösung auswählen und Akzeptieren klicken.



(a)

Das Datumsformat der Dateien ist abhängig von der auf der Internetseite eingestellten Sprache (Deutsch/English).

#### **53.6.1 NumPy**

```
dateipfad = "01-daten/Gro_handelspreise_202301010000_202401010000_Stunde.csv"
preise = np.loadtxt(fname = dateipfad, dtype = 'str', max_rows = 1)
preise
```

```
dateipfad = "01-daten/Gro_handelspreise_202301010000_202401010000_Stunde.csv"
preise = np.loadtxt(fname = dateipfad, dtype = 'str', max_rows = 1, delimiter = ';', encoding='utf-8')
preise
```

```
preise = np.loadtxt(fname = dateipfad, dtype = 'str', max_rows = 2, delimiter = ';', encoding = 'utf-8')
preise
```

```

preise = np.loadtxt(fname = dateipfad, dtype = 'str', delimiter = ';', encoding = 'UTF-8-sig'

# Datumsspalten isolieren
preise_date = preise[ : , 0:2]

# Zeichenkette manuell ins Format ISO 8601 bringen
## Spalte 0
### neues Array anlegen
preise_datumvon = np.array([], dtype = 'datetime64')

for element in preise_date[ : , 0]:

    # string umstellen
    neues_element = element[6:10] + '-' + \
    element[3:5] + '-' + \
    element[0:2] + 'T' + \
    element[11:13] + ':' + \
    element[14:]

    # in datetime64 konvertieren
    neues_element = np.datetime64(neues_element)

    # anhängen
    preise_datumvon = np.append(preise_datumvon, neues_element)

## Spalte 1
### neues Array anlegen
preise_datumbis = np.array([], dtype = 'datetime64')

for element in preise_date[ : , 1]:

    # string umstellen
    neues_element = element[6:10] + '-' + \
    element[3:5] + '-' + \
    element[0:2] + 'T' + \
    element[11:13] + ':' + \
    element[14:]

```

```
# in datetime64 konvertieren
neues_element = np.datetime64(neues_element)

# anhängen
preise_datumbis = np.append(preise_datumbis, neues_element)

# die letzten 4 Elemente angucken
print(preise_datumvon[-4:], preise_datumvon.dtype)
print(preise_datumbis[-4:], preise_datumbis.dtype)
```

```
# numerische Spalten isolieren
preise_numeric = preise[ :, 2:]

# Position der Spalte mit fehlendem Wert '-' in der nullten Zeile finden
position = np.argwhere(preise_numeric[0, : ] == '-')
print("Spaltenindex:", position)

# prüfen, welche Werte in der Spalte vorkommen
print("Anzahl einzigartiger Werte:", len(np.unique(preise_numeric[:, position])))
```

```
# Spalte mit fehlenden Werten entfernen
preise_numeric = np.delete(arr = preise_numeric, obj = position, axis = 1) # axis 1 = columns
```

```
# Dezimaltrennzeichen ersetzen
preise_numeric = np.char.replace(preise_numeric, ',', '.')
preise_numeric = preise_numeric.astype('float64')

# Spaltennamen speichern
preise_numeric_colnames = np.loadtxt(fname = dateipfad, dtype = 'str', delimiter = ';', encoding = 'utf-8')
preise_numeric_colnames = preise_numeric_colnames[2:] # Datumsspalten entfernen
preise_numeric_colnames = np.delete(arr = preise_numeric_colnames, obj = position)

print(preise_numeric_colnames, "\n")
print(preise_numeric[0:2, :], preise_numeric.dtype)
```

### 53.6.2 Pandas

```
dateipfad = "01-daten/Gro_handelspreise_202301010000_202401010000_Stunde.csv"
preise = pd.read_csv(filepath_or_buffer = dateipfad)
print(preise.info())
```

```
dateipfad = "01-daten/Gro_handelspreise_202301010000_202401010000_Stunde.csv"
preise = pd.read_csv(filepath_or_buffer = dateipfad, sep = ';')
print(preise.info())
preise.head()
```

```
preise = pd.read_csv(filepath_or_buffer = dateipfad, sep = ';', decimal = ',')  
print(preise.info())
```

```
preise['DE/AT/LU [€/MWh] Originalauflösungen'].describe()
```

Dokumentation

strftime-

```
preise.drop(labels = 'DE/AT/LU [€/MWh] Originalauflösungen', axis = 'columns', inplace = True)

## Datumsspalten konvertieren
preise['Datum von'] = pd.to_datetime(preise['Datum von'], format = "%d.%m.%Y %H:%M")
preise['Datum bis'] = pd.to_datetime(preise['Datum bis'], format = "%d.%m.%Y %H:%M")
print(preise.info())
```

```
preise = pd.read_csv(filepath_or_buffer = dateipfad, sep = ';', decimal = ',',
                     usecols = list(range(0, 15)) + list(range(16, 19)), # Auswahl der einzuh
                     parse_dates = ['Datum von', 'Datum bis'], date_format = "%d.%m.%Y %H:%M"
print(preise.info())
```

## 53.7 Zugriff auf Zeitreihen

```
# Attribute
print("Jahr:", pd.to_datetime(0).year)
print("Monat:", pd.to_datetime(0).month)
print("Tag:", pd.to_datetime(0).day)
print("Stunde:", pd.to_datetime(0).hour)
print("Minute:", pd.to_datetime(0).minute)
print("Sekunde:", pd.to_datetime(0).second)
print("Tag des Jahres:", pd.to_datetime(0).dayofyear)
print("Wochentag:", pd.to_datetime(0).dayofweek)
print("Tage im Monat:", pd.to_datetime(0).days_in_month)
print("Schaltjahr:", pd.to_datetime(0).is_leap_year)

# Methoden
print("\nDatum:", pd.to_datetime(0).date())
```

```
print("Zeit:", pd.to_datetime(0).time())
print("Wochentag (0-6):", pd.to_datetime(0).weekday())
print("Monatsname:", pd.to_datetime(0).month_name())
```

.dt accessor

## Der dt-Operator

```
# Attribute
print("Datum:", pd.Series(pd.to_datetime(0)).dt.date) # Unterschied
print("Zeit:", pd.Series(pd.to_datetime(0)).dt.time) # Unterschied
print("Jahr", pd.Series(pd.to_datetime(0)).dt.year)
print("Monat", pd.Series(pd.to_datetime(0)).dt.month)
print("Tag", pd.Series(pd.to_datetime(0)).dt.day)
print("Stunde", pd.Series(pd.to_datetime(0)).dt.hour)
print("Minute", pd.Series(pd.to_datetime(0)).dt.minute)
print("Sekunde", pd.Series(pd.to_datetime(0)).dt.second)

print("\nTag des Jahres", pd.Series(pd.to_datetime(0)).dt.dayofyear)
print("Wochentag:", pd.Series(pd.to_datetime(0)).dt.dayofweek)
print("Wochentag:", pd.Series(pd.to_datetime(0)).dt.weekday) # Unterschied
print("Tage im Monat:", pd.Series(pd.to_datetime(0)).dt.days_in_month)
print("Schaltjahr:", pd.Series(pd.to_datetime(0)).dt.is_leap_year)

# Methoden
print("\nName des Monats:", pd.Series(pd.to_datetime(0)).dt.month_name())
```

Datum: 0 1970-01-01  
dtype: object  
Zeit: 0 00:00:00  
dtype: object  
Jahr 0 1970  
dtype: int32  
Monat 0 1  
dtype: int32  
Tag 0 1  
dtype: int32  
Stunde 0 0  
dtype: int32  
Minute 0 0  
dtype: int32  
Sekunde 0 0  
dtype: int32

Tag des Jahres 0 1  
dtype: int32  
Wochentag: 0 3  
dtype: int32

```

Wochentag: 0      3
dtype: int32
Tage im Monat: 0     31
dtype: int32
Schaltjahr: 0    False
dtype: bool

Name des Monats: 0    January
dtype: object

```

 Tip ???: Musterlösung Strompreisvergleich

```

## Zugriff mit .dt für pd.Series
# Werktag und Wochenende unterscheiden
werktags = preise['Datum von'].dt.weekday.isin(list(range(0, 5)))
wochenende = preise['Datum von'].dt.weekday.isin(list(range(5, 7)))

print(werktags.head())
print(wochenende.head())

# Preise vergleichen
preis_werktags = preise.loc[werktags, 'Deutschland/Luxemburg [€/MWh] Originalauflösungen']
preis_wochenende = preise.loc[wochenende, 'Deutschland/Luxemburg [€/MWh] Originalauflösungen']

print(f"\nDurchschnittspreis werktags: {preis_werktags:.2f} [€/MWh]\nDurchschnittspreis am
      Wochenende: {preis_wochenende:.2f} [€/MWh]\n")
      
```

```

0    False
1    False
2    False
3    False
4    False
Name: Datum von, dtype: bool
0    True
1    True
2    True
3    True
4    True

```

Name: Datum von, dtype: bool

Durchschnittspreis werktags: 103.18 [€/MWh]

Durchschnittspreis am Wochenende: 75.34 [€/MWh]

### 53.8 Fehlende Werte in Zeitreihen

- <https://numpy.org/doc/stable/reference/arrays.datetime.html>
- [https://pandas.pydata.org/docs/user\\_guide/missing\\_data.html](https://pandas.pydata.org/docs/user_guide/missing_data.html)

### ⚠ Warning ??: Achtung Logik!

Die logische Abfrage fehlender Werte unterscheidet sich für `None`, `np.nan` und `pd.NA` und `pd.NaT`.

```
bool_values = [None, float('nan'), pd.NA, pd.NaT]

for element in bool_values:
    try:
        bool_value = bool(element)
    except TypeError as error:
        print(error)
    else:
        print("Wahrheitswert von", element, "ist", bool_value)
```

```
Wahrheitswert von None ist False
Wahrheitswert von nan ist True
boolean value of NA is ambiguous
Wahrheitswert von NaT ist True
```

Dies gilt auch für die Wertgleichheit.

```
for element in bool_values:
    try:
        result = element == element
    except TypeError as error:
        print(error)
    else:
        print("Wertgleichheit von", element, "ist", result)
```

```
Wertgleichheit von None ist True
Wertgleichheit von nan ist False
Wertgleichheit von <NA> ist <NA>
Wertgleichheit von NaT ist False
```

## 53.9 Übungen: Zeitreihen einlesen

“everybody I know has war stories about cleaning up lousy datasets”

Nicholas J. Cox

Cox, Nicholas J. 2004: Exploratory Data Mining and Data Cleaning. Book Review 9. In: Journal of Statistical Software 2004, Volume 11. <https://www.jstatsoft.org/article/view/v011b09/>  
30

### 53.9.0.1 Leicht: Englisches Datumsformat einlesen

#### Dokumentation strftime zur string-Formatierung

💡 Tip ???: Musterlösung Strompreise

```
import pandas as pd

dateipfad = "01-daten/Day-ahead_prices_202301010000_202401010000_Hour.csv"

data = pd.read_csv(dateipfad, sep=";")    # Semikolon als Trennzeichen muss angegeben werden
data.info() # -> man sieht, dass Spalten 0 "Start date" und 1 "End date" als Dtype "object"
print("\n")

print(data.iloc[0:10, 0:2], "\n") # anzeigen von ein paar Zeilen, um zu schauen wie die ersten
# Ausgabe lautet:
# Start date: Jan 1, 2023 12:00 AM
# End date Jan 1, 2023 1:00 AM
# also bieten sich zwei Varianten an:

# 1. Variante: beim Einlesen schon Datumsformat angeben:
data = pd.read_csv(dateipfad, sep=";" , parse_dates=[0,1], date_format="%b %d, %Y %I:%M %p")

# 2. Variante: Die beiden Spalten zu Anfang, die dtype object haben, separat ändern nachdem
data["Start date"] = pd.to_datetime(data["Start date"], format="%b %d, %Y %I:%M %p")    #%b %d, %Y %I:%M %p"
data["End date"] = pd.to_datetime(data["End date"], format="%b %d, %Y %I:%M %p")
print(data.iloc[0:10, 0:2], "\n")

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 19 columns):
 #   Column                                         Non-Null Count  Dtype 

```

```

0 Start date 8760 non-null object
1 End date 8760 non-null object
2 Germany/Luxembourg [€/MWh] Original resolutions 8760 non-null float64
3 DE/LU neighbours [€/MWh] Original resolutions 8760 non-null float64
4 Belgium [€/MWh] Original resolutions 8760 non-null float64
5 Denmark 1 [€/MWh] Original resolutions 8760 non-null float64
6 Denmark 2 [€/MWh] Original resolutions 8760 non-null float64
7 France [€/MWh] Original resolutions 8760 non-null float64
8 Netherlands [€/MWh] Original resolutions 8760 non-null float64
9 Norway 2 [€/MWh] Original resolutions 8760 non-null float64
10 Austria [€/MWh] Original resolutions 8760 non-null float64
11 Poland [€/MWh] Original resolutions 8760 non-null float64
12 Sweden 4 [€/MWh] Original resolutions 8760 non-null float64
13 Switzerland [€/MWh] Original resolutions 8760 non-null float64
14 Czech Republic [€/MWh] Original resolutions 8760 non-null float64
15 DE/AT/LU [€/MWh] Original resolutions 8760 non-null object
16 Northern Italy [€/MWh] Original resolutions 8760 non-null float64
17 Slovenia [€/MWh] Original resolutions 8760 non-null float64
18 Hungary [€/MWh] Original resolutions 8760 non-null float64
dtypes: float64(16), object(3)
memory usage: 1.3+ MB

```

	Start date	End date
0	Jan 1, 2023 12:00 AM	Jan 1, 2023 1:00 AM
1	Jan 1, 2023 1:00 AM	Jan 1, 2023 2:00 AM
2	Jan 1, 2023 2:00 AM	Jan 1, 2023 3:00 AM
3	Jan 1, 2023 3:00 AM	Jan 1, 2023 4:00 AM
4	Jan 1, 2023 4:00 AM	Jan 1, 2023 5:00 AM
5	Jan 1, 2023 5:00 AM	Jan 1, 2023 6:00 AM
6	Jan 1, 2023 6:00 AM	Jan 1, 2023 7:00 AM
7	Jan 1, 2023 7:00 AM	Jan 1, 2023 8:00 AM
8	Jan 1, 2023 8:00 AM	Jan 1, 2023 9:00 AM
9	Jan 1, 2023 9:00 AM	Jan 1, 2023 10:00 AM

	Start date	End date
0	2023-01-01 00:00:00	2023-01-01 01:00:00
1	2023-01-01 01:00:00	2023-01-01 02:00:00
2	2023-01-01 02:00:00	2023-01-01 03:00:00
3	2023-01-01 03:00:00	2023-01-01 04:00:00
4	2023-01-01 04:00:00	2023-01-01 05:00:00

```

5 2023-01-01 05:00:00 2023-01-01 06:00:00
6 2023-01-01 06:00:00 2023-01-01 07:00:00
7 2023-01-01 07:00:00 2023-01-01 08:00:00
8 2023-01-01 08:00:00 2023-01-01 09:00:00
9 2023-01-01 09:00:00 2023-01-01 10:00:00

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8760 entries, 0 to 8759
Data columns (total 19 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Start date      8760 non-null    datetime64[ns]
 1   End date        8760 non-null    datetime64[ns]
 2   Germany/Luxembourg [€/MWh] Original resolutions 8760 non-null    float64 
 3   DE/LU neighbours [€/MWh] Original resolutions 8760 non-null    float64 
 4   Belgium [€/MWh] Original resolutions 8760 non-null    float64 
 5   Denmark 1 [€/MWh] Original resolutions 8760 non-null    float64 
 6   Denmark 2 [€/MWh] Original resolutions 8760 non-null    float64 
 7   France [€/MWh] Original resolutions 8760 non-null    float64 
 8   Netherlands [€/MWh] Original resolutions 8760 non-null    float64 
 9   Norway 2 [€/MWh] Original resolutions 8760 non-null    float64 
 10  Austria [€/MWh] Original resolutions 8760 non-null    float64 
 11  Poland [€/MWh] Original resolutions 8760 non-null    float64 
 12  Sweden 4 [€/MWh] Original resolutions 8760 non-null    float64 
 13  Switzerland [€/MWh] Original resolutions 8760 non-null    float64 
 14  Czech Republic [€/MWh] Original resolutions 8760 non-null    float64 
 15  DE/AT/LU [€/MWh] Original resolutions 8760 non-null    object  
 16  Northern Italy [€/MWh] Original resolutions 8760 non-null    float64 
 17  Slovenia [€/MWh] Original resolutions 8760 non-null    float64 
 18  Hungary [€/MWh] Original resolutions 8760 non-null    float64 
dtypes: datetime64[ns](2), float64(16), object(1)
memory usage: 1.3+ MB

```

Musterlösung von Marc Sönnecken. Zur Verbesserung der Lesbarkeit wurde die Ausgabe mit `print(data.head(10))` ersetzt durch `print(data.iloc[0:10, 0:2], "\n")`. Um sich einen Überblick über einen Datensatz zu verschaffen, ist die Methode `.head()` jedoch besser geeignet

### 53.9.0.2 Mittel: Strommarktdaten Österreich

<https://markttransparenzapg.at/>

#### ⚠ Warning ??: Markttransparenzdaten Österreich herunterladen

Nach der Auswahl des Zeitraums auf Exportieren klicken, dann erscheint die Schaltfläche Download.

The screenshot shows a user interface for data export. At the top, there are three tabs: 'Diagramm', 'Tabelle', and 'Export'. The 'Export' tab is selected. Below the tabs, there are four input fields: 'Datum von:' with value '01.01.2023', 'Datum bis:' with value '31.12.2023', 'Auflösung:' with value '15 Min.', and 'Jahr:' with value '2023'. Underneath these are dropdown menus for 'Monat:' and 'Wählen:', and two buttons: 'Exportieren' (highlighted in red) and 'Download'.

Figure 53.3:

The screenshot shows a user interface for data export. At the top, there are three tabs: 'Chart', 'Table', and 'Export'. The 'Table' tab is selected. Below the tabs, there are four input fields: 'Date from:' with value '01/01/2023', 'Date to:' with value '12/31/2023', 'Resolution:' with value '15 Min.', and 'Year:' with value '2023'. Underneath are dropdown menus for 'Month:' and 'Choose', and two buttons: 'Export' (highlighted in red) and 'Download'.

Figure 53.4:

Das Datumsformat der Dateien ist abhängig von der auf der Internetseite eingestellten Sprache (Deutsch/English).

	A	B	C	D
1	Zeit von [CET/CEST]	Zeit bis [CET/CEST]	Wind [MW]	Solar [MW]
28899	29.10.2023 01:15:00	29.10.2023 01:30:00	396	0
28900	29.10.2023 01:30:00	29.10.2023 01:45:00	360	0
28901	29.10.2023 01:45:00	29.10.2023 2A:00:00	308	0
28902	29.10.2023 2A:00:00	29.10.2023 2A:15:00	300	0
28903	29.10.2023 2A:15:00	29.10.2023 2A:30:00	296	0
28904	29.10.2023 2A:30:00	29.10.2023 2A:45:00	288	0
28905	29.10.2023 2A:45:00	29.10.2023 2B:00:00	272	0
28906	29.10.2023 2B:00:00	29.10.2023 2B:15:00	264	0
28907	29.10.2023 2B:15:00	29.10.2023 2B:30:00	260	0
28908	29.10.2023 2B:30:00	29.10.2023 2B:45:00	252	0
28909	29.10.2023 2B:45:00	29.10.2023 03:00:00	240	0
28910	29.10.2023 03:00:00	29.10.2023 03:15:00	244	0

Figure 53.5: Zeitumstellung im österreichischen Datensatz

💡 Tip ???: Musterlösung Strommarktdaten Österreich

### einfache Variante

Die einfachste Lösung ist es, Zeitreihen zu generieren und die Spalten ‘Zeit von [CET/CEST]’ und ‘Zeit bis [CET/CEST]’ damit zu ersetzen.

```
von = pd.date_range(start = "2023-01-01T00:00", end = "2023-12-31T23:45", freq = '15min')
bis = pd.date_range(start = "2023-01-01T00:15", end = "2024-01-01T00:00", freq = '15min')

print(von[8070:8078], "\n")
print(bis[8070:8078], "\n")
```

### Datei einlesen und String-Manipulation

Zunächst wird die Datei eingelesen. Die Zellen, die sich nicht in datetime umwandeln lassen, können mit Python ausgegeben werden.

```

# Datei einlesen
erzeugung_austria = pd.read_csv(filepath_or_buffer = "01-daten/AGPT_2022-12-31T23_00_00Z_2023-10-29.csv",
                                 sep = ";", decimal = ",", thousands = ".")

# Zellen mit fehlerhaften datetime strings identifizieren
print("Spalte 'Zeit von [CET/CEST]'")
i = 0
position_element = []
for element in erzeugung_austria['Zeit von [CET/CEST]']:
    try:
        pd.to_datetime(element, format = "%d.%m.%Y %H:%M:%S")
    except:
        print(element)
        position_element.append(i)
    i += 1
print("\nDie Zellen haben den Zeilenindex: ", position_element, "\n")

print("Spalte 'Zeit bis [CET/CEST]'")
i = 0
position_element = []
for element in erzeugung_austria['Zeit bis [CET/CEST]']:
    try:
        pd.to_datetime(element, format = "%d.%m.%Y %H:%M:%S")
    except:
        print(element)
        position_element.append(i)
    i += 1
print("\nDie Zellen haben den Zeilenindex: ", position_element, "\n")

```

Spalte 'Zeit von [CET/CEST]'

29.10.2023 2A:00:00  
 29.10.2023 2A:15:00  
 29.10.2023 2A:30:00  
 29.10.2023 2A:45:00  
 29.10.2023 2B:00:00  
 29.10.2023 2B:15:00  
 29.10.2023 2B:30:00  
 29.10.2023 2B:45:00

Die Zellen haben den Zeilenindex: [28900, 28901, 28902, 28903, 28904, 28905, 28906, 28907]

```
Spalte 'Zeit bis [CET/CEST]'  
29.10.2023 2A:00:00  
29.10.2023 2A:15:00  
29.10.2023 2A:30:00  
29.10.2023 2A:45:00  
29.10.2023 2B:00:00  
29.10.2023 2B:15:00  
29.10.2023 2B:30:00  
29.10.2023 2B:45:00
```

Die Zellen haben den Zeilenindex: [28899, 28900, 28901, 28902, 28903, 28904, 28905, 28906]

Damit die Datumsspalten korrekt eingelesen werden können, werden die Zeichenfolgen "2A" und "2B" mit der Methode `str.replace()` durch "02" ersetzt. Dadurch wird eine Dublette im Datensatz erzeugt.

```
# string replace & als Datum einlesen  
## Spalte Zeit von [CET/CEST]  
erzeugung_austria['Zeit von [CET/CEST]'] = erzeugung_austria['Zeit von [CET/CEST]'].str.replace('2A', '02')  
erzeugung_austria['Zeit von [CET/CEST]'] = erzeugung_austria['Zeit von [CET/CEST]'].str.replace('2B', '02')  
  
erzeugung_austria['Zeit von [CET/CEST]'] = pd.to_datetime(erzeugung_austria['Zeit von [CET/CEST]'])  
  
## Spalte Zeit bis [CET/CEST]  
erzeugung_austria['Zeit bis [CET/CEST]'] = erzeugung_austria['Zeit bis [CET/CEST]'].str.replace('2A', '02')  
erzeugung_austria['Zeit bis [CET/CEST]'] = erzeugung_austria['Zeit bis [CET/CEST]'].str.replace('2B', '02')  
  
erzeugung_austria['Zeit bis [CET/CEST]'] = pd.to_datetime(erzeugung_austria['Zeit bis [CET/CEST]'])  
  
print(erzeugung_austria.info())  
  
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 35040 entries, 0 to 35039  
Data columns (total 15 columns):  
 #   Column           Non-Null Count  Dtype     
 ---  --  
 0   Zeit von [CET/CEST]    35040 non-null   datetime64[ns]  
 1   Zeit bis [CET/CEST]    35040 non-null   datetime64[ns]  
 2   Wind [MW]             35040 non-null   float64  
 3   Solar [MW]            35040 non-null   float64  
 4   Biomasse [MW]         35040 non-null   float64  
 5   Gas [MW]              35040 non-null   float64
```

```
6    Kohle [MW]                35040 non-null float64
7    Öl [MW]                   35040 non-null float64
8    Geothermie [MW]           35040 non-null float64
9    Pumpspeicher [MW]         35040 non-null float64
10   Lauf- und Schwellwasser [MW] 35040 non-null float64
11   Speicher [MW]             35040 non-null float64
12   Sonstige Erneuerbare [MW]  35040 non-null float64
13   Müll [MW]                 35040 non-null float64
14   Andere [MW]               35040 non-null float64
dtypes: datetime64[ns](2), float64(13)
memory usage: 4.0 MB
None
```

### Indexpositionen der doppelten und der fehlenden Stunde bestimmen

Im nächsten Schritt werden die Indexpositionen der doppelten und der fehlenden Stunde bestimmt. Dazu wird ein neues Objekt angelegt, das auf den Speicherbereich der Datums-Spalten zugreift (was nicht zwingend erforderlich ist). Die Position der doppelten Stunde wird mit der Methode `pd.Series.duplicated()` bestimmt, die einen logischen Vektor zurückgibt. Dieser wird zum Slicing und der Ausgabe der Indexpositionen verwendet. Durch die Subtraktion von 4 wird der Index der ersten Stunde ausgegeben (der Datensatz ist auf Viertelstundenbasis).

*doppelte Stunde*

```

# neues Objekt anlegen
austria_dates = erzeugung_austria[['Zeit von [CET/CEST]', 'Zeit bis [CET/CEST]']].copy()

# Indexposition der doppelten Stunde bestimmen
## Zeit von
position_doppelte_stunde_von = austria_dates['Zeit von [CET/CEST]'][austria_dates['Zeit von [CET/CEST]'] == '2023-10-29 02:45:00'].index[0]

print(f"Die doppelte Stunde (Zeit von):\n{austria_dates.loc[position_doppelte_stunde_von, 'Zeit von [CET/CEST]']}  
f"\nDie nächste Stunde lautet:\n{austria_dates.loc[position_doppelte_stunde_von + 4, 'Zeit von [CET/CEST]']}  
f"\n\nBeide Stunden sind identisch.")

### Ende der Verschiebung in Spalte Zeit von
ende_verschiebung_von = position_doppelte_stunde_von[-1]
print(f"\nDie Zeitverschiebung in der Spalte Zeit von endet bei Indexposition: {ende_verschiebung_von}")

## Zeit bis
position_doppelte_stunde_bis = austria_dates['Zeit bis [CET/CEST]'][austria_dates['Zeit bis [CET/CEST]'] == '2023-10-29 02:45:00'].index[0]

print(f"\n\nDie doppelte Stunde (Zeit bis):\n{austria_dates.loc[position_doppelte_stunde_bis, 'Zeit bis [CET/CEST]']}  
f"\nDie nächste Stunde lautet:\n{austria_dates.loc[position_doppelte_stunde_bis + 4, 'Zeit bis [CET/CEST]']}  
f"\n\nBeide Stunden sind identisch.")

### Ende der Verschiebung in Spalte Zeit bis
ende_verschiebung_bis = position_doppelte_stunde_bis[-1]
print(f"\nDie Zeitverschiebung in der Spalte Zeit bis endet bei Indexposition: {ende_verschiebung_bis}")

Die doppelte Stunde (Zeit von):
28900 2023-10-29 02:00:00
28901 2023-10-29 02:15:00
28902 2023-10-29 02:30:00
28903 2023-10-29 02:45:00
Name: Zeit von [CET/CEST], dtype: datetime64[ns]
steht an Indexposition
Index([28900, 28901, 28902, 28903], dtype='int64')

Die nächste Stunde lautet:
28904 2023-10-29 02:00:00
28905 2023-10-29 02:15:00
28906 2023-10-29 02:30:00
28907 2023-10-29 02:45:00
Name: Zeit von [CET/CEST], dtype: datetime64[ns]

```

Beide Stunden sind identisch.

Die Zeitverschiebung in der Spalte Zeit von endet bei Indexposition: 28903

Die doppelte Stunde (Zeit bis):

```
28899 2023-10-29 02:00:00  
28900 2023-10-29 02:15:00  
28901 2023-10-29 02:30:00  
28902 2023-10-29 02:45:00
```

Name: Zeit bis [CET/CEST], dtype: datetime64[ns]

steht an Indexposition

```
Index([28899, 28900, 28901, 28902], dtype='int64')
```

Die nächste Stunde lautet:

```
28903 2023-10-29 02:00:00  
28904 2023-10-29 02:15:00  
28905 2023-10-29 02:30:00  
28906 2023-10-29 02:45:00
```

Name: Zeit bis [CET/CEST], dtype: datetime64[ns]

Beide Stunden sind identisch.

Die Zeitverschiebung in der Spalte Zeit bis endet bei Indexposition: 28902

#### *fehlende Stunde*

Die Sommerzeit beginnt am letzten Sonntag im März. Die Stunde liegt nicht in range(0, 24). Diese Bedingung kann in vier Schritten kontrolliert werden:

- Monat März: `march = pd.Series[pd.Series.dt.month == 3]`
- Sonntage im März: `sundays = march[march.dt.dayofweek == 6]`
- letzter Sonntag im März: Die letzten  $23*4$  Einträge sind der letzte Sonntag des Monats (23 weil eine Stunde fehlt).  
`last_sunday = sundays[-23*4:]`
- fehlende Stunde: `np.argwhere(np.invert(pd.Series(range(0,24)).isin(last_sunday.dt.hour)))`

```

# Indexposition der fehlenden Stunde bestimmen
## Zeit von
### Monat März
maske_märz_von = austria_dates['Zeit von [CET/CEST]'].dt.month == 3
austria_dates_march_von = austria_dates.loc[maske_märz_von, 'Zeit von [CET/CEST]']
print(f"Der Monat März (Zeit von):\n{austria_dates_march_von.head}\n");

### letzter Sonntag
maske_sonntag_von = (austria_dates_march_von.dt.dayofweek == 6)
letzter_sonntag_von = (austria_dates_march_von[maske_sonntag_von]) [-23*4 :]
print(f"Der letzte Sonntag im März:\n{letzter_sonntag_von}\n")

### fehlende Stunde
print(letzter_sonntag_von.dt.hour)
fehlende_stunde_von = np.argwhere(np.invert(pd.Series(range(0,24)).isin(letzter_sonntag_von)))
print(f"\nEs fehlt die Stunde:\n{fehlende_stunde_von}\n")
print(letzter_sonntag_von[letzter_sonntag_von.dt.hour == (fehlende_stunde_von - 1)], letzte

### Beginn der Verschiebung in Spalte Zeit von
beginn_verschiebung_von = letzter_sonntag_von[letzter_sonntag_von.dt.hour == (fehlende_stunde_von + 1)]
print(f"\nDie Zeitverschiebung in der Spalte Zeit von beginnt bei Indexposition: {beginn_von}\n")

## Zeit bis
### Monat März
maske_märz_bis = austria_dates['Zeit bis [CET/CEST]'].dt.month == 3
austria_dates_march_bis = austria_dates.loc[maske_märz_bis, 'Zeit bis [CET/CEST]']
print(f"Der Monat März (Zeit bis):\n{austria_dates_march_bis.head}\n");

### letzter Sonntag
maske_sonntag_bis = (austria_dates_march_bis.dt.dayofweek == 6)
letzter_sonntag_bis = (austria_dates_march_bis[maske_sonntag_bis]) [-23*4 :]
print(f"Der letzte Sonntag im März:\n{letzter_sonntag_bis}\n")

### fehlende Stunde
print(letzter_sonntag_bis.dt.hour)
fehlende_stunde_bis = np.argwhere(np.invert(pd.Series(range(0,24)).isin(letzter_sonntag_bis)))
print(f"\nEs fehlt die Stunde:\n{fehlende_stunde_bis}\n")
print(letzter_sonntag_bis[letzter_sonntag_bis.dt.hour == (fehlende_stunde_bis + 1)], letzte

### Beginn der Verschiebung in Spalte Zeit bis
beginn_verschiebung_bis = letzter_sonntag_bis[letzter_sonntag_bis.dt.hour == (fehlende_stunde_bis + 1)]
print(f"\nDie Zeitverschiebung in der Spalte Zeit bis beginnt bei Indexposition: {beginn_von}\n")

```

```
Der Monat März (Zeit von):
```

```
<bound method NDFrame.head of 5664    2023-03-01 00:00:00
5665    2023-03-01 00:15:00
5666    2023-03-01 00:30:00
5667    2023-03-01 00:45:00
5668    2023-03-01 01:00:00
...
8631    2023-03-31 22:45:00
8632    2023-03-31 23:00:00
8633    2023-03-31 23:15:00
8634    2023-03-31 23:30:00
8635    2023-03-31 23:45:00
Name: Zeit von [CET/CEST], Length: 2972, dtype: datetime64[ns]>
```

```
Der letzte Sonntag im März:
```

```
8064    2023-03-26 00:00:00
8065    2023-03-26 00:15:00
8066    2023-03-26 00:30:00
8067    2023-03-26 00:45:00
8068    2023-03-26 01:00:00
...
8151    2023-03-26 22:45:00
8152    2023-03-26 23:00:00
8153    2023-03-26 23:15:00
8154    2023-03-26 23:30:00
8155    2023-03-26 23:45:00
Name: Zeit von [CET/CEST], Length: 92, dtype: datetime64[ns]
```

```
8064    0
8065    0
8066    0
8067    0
8068    1
..
8151    22
8152    23
8153    23
8154    23
8155    23
```

```
Name: Zeit von [CET/CEST], Length: 92, dtype: int32
```

Es fehlt die Stunde:

2

```
8068    2023-03-26 01:00:00
8069    2023-03-26 01:15:00
8070    2023-03-26 01:30:00
8071    2023-03-26 01:45:00
Name: Zeit von [CET/CEST], dtype: datetime64[ns]
8072    2023-03-26 03:00:00
8073    2023-03-26 03:15:00
8074    2023-03-26 03:30:00
8075    2023-03-26 03:45:00
Name: Zeit von [CET/CEST], dtype: datetime64[ns]
```

Die Zeitverschiebung in der Spalte Zeit von beginnt bei Indexposition: 8072

Der Monat März (Zeit bis):

```
<bound method NDFrame.head of 5663    2023-03-01 00:00:00
5664    2023-03-01 00:15:00
5665    2023-03-01 00:30:00
5666    2023-03-01 00:45:00
5667    2023-03-01 01:00:00
...
8630    2023-03-31 22:45:00
8631    2023-03-31 23:00:00
8632    2023-03-31 23:15:00
8633    2023-03-31 23:30:00
8634    2023-03-31 23:45:00
Name: Zeit bis [CET/CEST], Length: 2972, dtype: datetime64[ns]>
```

Der letzte Sonntag im März:

```
8063    2023-03-26 00:00:00
8064    2023-03-26 00:15:00
8065    2023-03-26 00:30:00
8066    2023-03-26 00:45:00
8067    2023-03-26 01:00:00
...
8150    2023-03-26 22:45:00
8151    2023-03-26 23:00:00
8152    2023-03-26 23:15:00
```

```
8153 2023-03-26 23:30:00
8154 2023-03-26 23:45:00
Name: Zeit bis [CET/CEST], Length: 92, dtype: datetime64[ns]
```

```
8063    0
8064    0
8065    0
8066    0
8067    1
...
8150    22
8151    23
8152    23
8153    23
8154    23
Name: Zeit bis [CET/CEST], Length: 92, dtype: int32
```

Es fehlt die Stunde:

2

```
8067 2023-03-26 01:00:00
8068 2023-03-26 01:15:00
8069 2023-03-26 01:30:00
8070 2023-03-26 01:45:00
Name: Zeit bis [CET/CEST], dtype: datetime64[ns]
8071 2023-03-26 03:00:00
8072 2023-03-26 03:15:00
8073 2023-03-26 03:30:00
8074 2023-03-26 03:45:00
Name: Zeit bis [CET/CEST], dtype: datetime64[ns]
```

Die Zeitverschiebung in der Spalte Zeit bis beginnt bei Indexposition: 8071

Mit den gespeicherten Indexpositionen können die betreffenden Zeitstempel verschoben werden:

- Spalte Zeit von: 8072 (Objekt beginn\_verschiebung\_von) bis 28903 (Objekt ende\_verschiebung\_von)
- Spalte Zeit bis: 8071 (Objekt beginn\_verschiebung\_bis) bis 28902 (Objekt ende\_verschiebung\_bis)

Für das Slicing wird die Methode `pd.Series.iloc[]` verwendet, die exklusiv index-

iert, d. h. die Endpositionen müssen um 1 erhöht werden. Durch Subtraktion von pd.Timedelta(1, unit = 'h') wird die Zeitverschiebung aus dem Datensatz entfernt.

```
# Zeitverschiebung korrigieren
## Zeit von
austria_dates['Zeit von [CET/CEST]'].iloc[beginn_verschiebung_von : ende_verschiebung_von + 1] = erzeugung_austria['Zeit von [CET/CEST]'] - pd.Timedelta(1, unit='h')

## Zeit bis
austria_dates['Zeit bis [CET/CEST]'].iloc[beginn_verschiebung_bis : ende_verschiebung_bis - 1] = erzeugung_austria['Zeit bis [CET/CEST]'] - pd.Timedelta(1, unit='h')

# Kontrolle
print("Kontrolle im Datensatz +/- eine Viertelstunde\n")
print("Die Spalte Zeit von")
print(erzeugung_austria['Zeit von [CET/CEST]'].iloc[beginn_verschiebung_von -1 : ende_verschiebung_von + 1])
print("Die Spalte Zeit bis")
print(erzeugung_austria['Zeit bis [CET/CEST]'].iloc[beginn_verschiebung_bis -1 : ende_verschiebung_bis + 1])
```

Kontrolle im Datensatz +/- eine Viertelstunde

Die Spalte Zeit von

```
8071    2023-03-26 01:45:00
8072    2023-03-26 02:00:00
8073    2023-03-26 02:15:00
8074    2023-03-26 02:30:00
8075    2023-03-26 02:45:00
...
28900   2023-10-29 01:00:00
28901   2023-10-29 01:15:00
28902   2023-10-29 01:30:00
28903   2023-10-29 01:45:00
28904   2023-10-29 02:00:00
```

Name: Zeit von [CET/CEST], Length: 20834, dtype: datetime64[ns]

Die Spalte Zeit bis

```
8070    2023-03-26 01:45:00
8071    2023-03-26 02:00:00
8072    2023-03-26 02:15:00
```

```
8073    2023-03-26 02:30:00
8074    2023-03-26 02:45:00
...
28899   2023-10-29 01:00:00
28900   2023-10-29 01:15:00
28901   2023-10-29 01:30:00
28902   2023-10-29 01:45:00
28903   2023-10-29 02:00:00
Name: Zeit bis [CET/CEST], Length: 20834, dtype: datetime64[ns]
```

### 53.9.0.3 Schwer: CAPE Ratio - ein Datensatz voller Tücken

[zur XLS-Datei](#)

[Webseite](#)

[Direktlink](#)

#### 💡 Tip ??: Hinweise und Musterlösung Shiller data

Schauen Sie sich den Datensatz zunächst mit einem Tabellenkalkulationsprogramm an. Bemerkenswerte Auffälligkeiten sind:

- Metadaten in den Zeilen 2 und 3, in denen teilweise auch Spaltenbeschriftungen eingetragen sind, sowie am Ende des Datensatzes,
- mehrzeilige Spaltenbeschriftungen,
- Leerspalten P und N,
- Kennzeichnung fehlender Werte durch 'NA' und leere Zellen '' sowie
- abweichende Formatierung des Monats Oktober in der Spalte Date 'YYYY-M'.

#### Tip 53.7: Lösungshilfe

Aufgrund der zahlreichen Auffälligkeiten ist es hilfreich, die Daten und die Kopfzeilen getrennt einzulesen. Den korrekten Zeilenindex können Sie entweder der ersten Betrachtung mit einem Tabellenkalkulationsprogramm entnehmen oder indem Sie einfach die ersten 10 oder 20 Zeilen des Datensatzes in Python einlesen. Dadurch können die Daten einfacher überblickt und mit Methoden der String-Bearbeitung manipuliert werden. In der Praxis ist es einfacher, die Spaltenbeschriftungen manuell mit dem Argument `names = Sequence of column labels to apply` einzutragen bzw. dies mit Hilfe eines Tabellenkalkulationsprogramms zu erledigen.

Außerdem empfiehlt es sich, schrittweise vorzugehen und für jedes Problem eine separate Lösung, mit Ausschnitten des Datensatzes bzw. mit dafür generierten Testdaten, zu entwickeln.

#### **Tip 53.6: Vollständige Musterlösung**

Zum Einlesen wird die Funktion `pd.read_excel()` verwendet. Mit dem Argument `sheet_name` kann das Tabellenblatt Data ausgewählt werden. Über die Methode `pd.head(n = 10)` kann der Zeilenindex bestimmt werden, an dem der Tabellenkopf endet und die Daten beginnen. Es handelt sich um die achte Zeile, die in Python den Zeilenindex 7 hat.

#### **Kopf einlesen**

```
dateipfad = '01-daten/shiller_data.xls'
shiller = pd.read_excel(io = dateipfad, sheet_name = 'Data')

# manuell Ende des Kopfs und Beginn der Daten identifizieren (auskommentiert)
# print(shiller.head(n = 10), "\n")

# Kopf einlesen
shiller_head = pd.read_excel(io = dateipfad, sheet_name = 'Data', skiprows = 1, n
print(shiller_head, "\n")
print(shiller_head.info()) # die leeren Spalten werden als numerisch erkannt
```

	0	1	2	\
0	Stock Market Data Used in "Irrational Exuberan...	NaN	NaN	
1	Robert J. Shiller	NaN	NaN	
2		NaN	NaN	
3		NaN	NaN	
4		NaN	S&P	
5		NaN	Comp.	Dividend
6	Date	P		D

	3	4	5	6	7	8	9	...	\
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	
3	NaN	Consumer	NaN	NaN	NaN	NaN	Real	...	
4	NaN	Price	NaN	Long	NaN	NaN	Total	...	
5	Earnings	Index	Date	Interest	Real	Real	Return	...	
6	E	CPI	Fraction	Rate GS10	Price	Dividend	Price	...	
	12	13		14	15	16	17	18	\
0	Cyclically	NaN	Cyclically	NaN	NaN	NaN	NaN	...	
1	Adjusted	NaN	Adjusted	NaN	NaN	NaN	NaN	...	
2	Price	NaN	Total	Return	Price	NaN	NaN	NaN	
3	Earnings	NaN		Earnings	NaN	NaN	Monthly	Real	
4	Ratio	NaN		Ratio	NaN	Excess	Total	Total	
5	P/E10 or	NaN	TR	P/E10 or	NaN	CAPE	Bond	Bond	
6	CAPE	NaN	TR	CAPE	NaN	Yield	Returns	Returns	
		19		20		21			
0		NaN		NaN		NaN			
1		NaN		NaN		NaN			
2		NaN		NaN		NaN			
3		NaN		NaN		NaN			
4		10 Year		10 Year		Real 10 Year			
5	Annualized	Stock	Annualized	Bonds	Excess	Annualized			
6	Real	Return	Real	Return		Returns			

[7 rows x 22 columns]

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 22 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   0       3 non-null      object 
 1   1       3 non-null      object 
 2   2       2 non-null      object 
 3   3       2 non-null      object 
 4   4       4 non-null      object 
 5   5       2 non-null      object 
 6   6       3 non-null      object
```

```
    7    7      2 non-null    object
    8    8      2 non-null    object
    9    9      4 non-null    object
   10   10     2 non-null    object
   11   11     4 non-null    object
   12   12     7 non-null    object
   13   13      0 non-null  float64
   14   14     7 non-null    object
   15   15      0 non-null  float64
   16   16     3 non-null    object
   17   17     4 non-null    object
   18   18     4 non-null    object
   19   19     3 non-null    object
   20   20     3 non-null    object
   21   21     3 non-null    object
dtypes: float64(2), object(20)
memory usage: 1.3+ KB
None
```

### Spaltenbeschriftungen isolieren

Anschließend kann der Kopf weiter bearbeitet werden, um die Spaltenbeschriftungen zu isolieren. Dafür gibt es verschiedene Möglichkeiten. Weitere Alternativen zur folgenden Variante finden Sie im nachfolgenden Beispiel. Spaltenweise erfolgt die Verkettung der Zeichenketten mit der Methode `pd.Series.str.cat()`, die nur für `pd.Series` verfügbar ist (weshalb mit einer Schleife die Spalten einzeln durchlaufen werden) und nur mit dem Datentyp ‘string’ verfügbar ist, was durch die Methode `astype('string')` sichergestellt wird.

Anschließend werden nicht zur Spaltenbeschriftung gehörende Zeichenketten mit der Methode `str.replace()` entfernt. Dabei erweist sich das Argument `regex = True` als nützlich.

```

# Spaltenbeschriftung mit Schleife erzeugen
shiller_column_labels = pd.Series()

for column in shiller_head:
    shiller_column_labels = pd.concat([shiller_column_labels, pd.Series(shiller_head[column])])

# Zeichenketten säubern
## erste Zelle entfernen Stock Market Data Used in "Irrational Exuberan...
shiller_column_labels = shiller_column_labels.astype('str').replace(shiller_head[0], '')

## 'Robert J. Shiller' entfernen
shiller_column_labels = shiller_column_labels.astype('str').replace(shiller_head[1], '')

## Leerzeichen entfernen
## regex = True um Leerzeichen innerhalb von Strings zu entfernen
shiller_column_labels = shiller_column_labels.astype('str').replace(' ', ' ', regex=True)

## sehr lange strings ersetzen
shiller_column_labels = shiller_column_labels.astype('str').replace('CyclicallyAdjustedRealPrice', 'RealPrice')
shiller_column_labels = shiller_column_labels.astype('str').replace('CyclicallyAdjustedRealDividend', 'RealDividend')

## Index zurücksetzen
shiller_column_labels.reset_index(inplace = True, drop = True)

print(shiller_column_labels)

```

0	Date
1	S&PComp.P
2	DividendD
3	EarningsE
4	ConsumerPriceIndexCPI
5	DateFraction
6	LongInterestRateGS10
7	RealPrice
8	RealDividend
9	RealTotalReturnPrice
10	RealEarnings
11	RealTRScaledEarnings
12	CAPE
13	
14	TRCAPE

```
15                               ExcessCAPEYield
16                               MonthlyTotalBondReturns
17                               RealTotalBondReturns
18           10YearAnnualizedStockRealReturn
19           10YearAnnualizedBondsRealReturn
20           Real10YearExcessAnnualizedReturns
21
dtype: object
```

Alternative Vorgehensweisen zur String-Manipulation des Dateikopfs

#### 53.9.0.4 Verwendung des NumPy-Datentyps 'str'

Die Angabe von `dtype = 'str'` führt zur Verwendung des NumPy-Datentyps `string` (`dtype = 'str'`), der veränderlich (mutable) ist. Nur damit funktioniert die Verkettung von Strings mit der Methode `PD.df.sum()`.

```
# Der NumPy-Datentyp string ist veränderlich
my_array = np.array([[1, 2], [3, 4]])
my_array[0] = ['a', 'b']
my_array

array([['a', 'b'],
       ['3', '4']], dtype='<U1')
```

Mit den Pandas-Datentypen 'string' und 'object' funktioniert das gezeigte Vorgehen nicht. Denn Pandas nutzt den Python-Datentyp 'string', der unveränderlich ist. Das bedeutet, es gibt keine Methoden, die eine angelegte Zeichenkette verändern können. Stattdessen geben Methoden wie `str.replace()` neue strings zurück.

```

## mit NumPy-Datentyp string
shiller_head = pd.read_excel(io = dateipfad, sheet_name = 'Data', skiprows = 1)

# Kopf mit NumPy-Datentyp string mit pd.sum() verketten
# print(shiller_head.astype('str').sum(skipna = True, axis = 0))

## Bereinigung des Datensatzes
### nan entfernen
shiller_head = shiller_head.astype('str').replace('nan', '')

### erste Zelle entfernen Stock Market Data Used in "Irrational Exuberan...
shiller_head = shiller_head.astype('str').replace(shiller_head.loc[0, 0], '')

### 'Robert J. Shiller' entfernen
shiller_head = shiller_head.astype('str').replace(shiller_head.loc[1, 0], '')

### Leerzeichen entfernen
### regex = True um Leerzeichen innerhalb von Strings zu entfernen
shiller_head = shiller_head.astype('str').replace(' ', '', regex = True)

### sehr lange strings ersetzen
shiller_head = shiller_head.astype('str').replace('CyclicallyAdjustedPriceEarn...
shiller_head = shiller_head.astype('str').replace('CyclicallyAdjustedTotalRetu...

### spaltenweise Zeilen verketten
shiller_head = shiller_head.astype('str').sum(skipna = True, axis = 0)

print("\nzusammengeführte Spaltennamen\n", shiller_head)

```

### 53.9.0.5 Verwendung von DF.agg() oder DF.apply()

Die Pandas-Methode `DF.agg()` aggregiert einen DataFrame zeilen- oder spaltenweise durch eine spezifizierbare Funktion. Die Pandas-Methode `DF.apply()` wendet eine Funktion zeilen- oder spaltenweise auf einen DataFrame an. Die Methoden machen also das selbe. Details zur Verwendung des [Lambda-Ausdrucks](#) und der Methode `join` aus der [Pythonbasis](#) finden Sie in den angegebenen Links.

```

shiller_head = pd.read_excel(io = dateipfad, sheet_name = 'Data', skiprows = 1)
# DF.agg()
print(shiller_head.agg(lambda x: ''.join(x.astype(str)))))

# DF.apply
print(shiller_head.apply(lambda x: ''.join(x.astype(str))))

```

### Daten einlesen

Beim Einlesen der Daten wird der Kopf kontrolliert sowie mit der Methode `.tail()` das Ende der Datenreihe bestimmt, an dem weitere Metadaten vermerkt sind. Diese Metadaten werden anschließend mit dem Argument `skipfooter = 1` übersprungen.

```
dateipfad = '01-daten/shiller_data.xls'
```

```
# Daten einlesen
```

```
shiller_data = pd.read_excel(io = dateipfad, sheet_name = 'Data', skiprows = 8, header = 1)
print(shiller_data.head(n = 2), "\n")
shiller_data.tail(n = 5)
```

	0	1	2	3	4	5	6	7	8	9	...	\
0	1871.01	4.44	0.26	0.4	12.46	1871.04	5.32	111.06	6.50	111.06	...	
1	1871.02	4.5	0.26	0.4	12.84	1871.12	5.32	109.22	6.31	109.75	...	

	12	13	14	15	16	17	18	19	20	21		
0	NaN	NaN	NaN	NaN	NaN	1.0	1.00	0.13	0.09	0.04		
1	NaN	NaN	NaN	NaN	NaN	1.0	0.97	0.13	0.09	0.04		

[2 rows x 22 columns]

0	1	2	3	4	5	6	7	8	9	...	12	13	14	15	16	17	18	19	20	21
1836	2024.01	4815.61									70.48	NaN	308.42						2024.04	4.06
1837	2024.02	5011.96									70.65	NaN	310.33						2024.12	4.21
1838	2024.03	5170.57									70.82	NaN	311.28						2024.21	4.21
1839	2024.04	5243.77									NaN	NaN	311.76						2024.29	4.2
1840	NaN	Apr price is Apr 1st close									NaN	NaN	Mar/Apr CPI estimated						Apr GS10 is	

Nachdem die Metadaten entfernt wurden, werden die erkannten Datentypen mit der Methode `.info()` kontrolliert.

```
shiller_data = pd.read_excel(io = dateipfad, sheet_name = 'Data', skiprows = 8, header = 1)
print(shiller_data.head(n = 2), "\n")
print(shiller_data.tail(n = 2))

# Datentypen bestimmen
shiller_data.info()

      0      1      2      3      4      5      6      7      8      9    ... \
0  1871.01  4.44  0.26  0.4  12.46  1871.04  5.32  111.06  6.50  111.06 ...
1  1871.02  4.50  0.26  0.4  12.84  1871.12  5.32  109.22  6.31  109.75 ...

      12     13     14     15     16     17     18     19     20     21
0  NaN  NaN  NaN  NaN  NaN  1.0  1.00  0.13  0.09  0.04
1  NaN  NaN  NaN  NaN  NaN  1.0  0.97  0.13  0.09  0.04

[2 rows x 22 columns]

      0      1      2      3      4      5      6      7      8    ... \
1838  2024.03  5170.57  70.82  NaN  311.28  2024.21  4.21  5178.50  70.93
1839  2024.04  5243.77  NaN  NaN  311.76  2024.29  4.20  5243.77  NaN

      9    ...   12   13   14   15   16   17   18   19   20   21
1838  3.43e+06  ...  34.02  NaN  36.79  NaN  0.02  1.0  39.27  NaN  NaN
1839  3.48e+06  ...  34.41  NaN  37.18  NaN  0.01  NaN  39.37  NaN  NaN

[2 rows x 22 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1840 entries, 0 to 1839
Data columns (total 22 columns):
 #   Column  Non-Null Count  Dtype  
--- 
 0   0        1840 non-null   float64
 1   1        1840 non-null   float64
 2   2        1839 non-null   float64
 3   3        1836 non-null   float64
 4   4        1840 non-null   float64
 5   5        1840 non-null   float64
 6   6        1840 non-null   float64
 7   7        1840 non-null   float64
```

```

8    8      1839 non-null   float64
9    9      1840 non-null   float64
10   10     1836 non-null   float64
11   11     1836 non-null   float64
12   12     1720 non-null   float64
13   13      0 non-null    float64
14   14     1720 non-null   float64
15   15      0 non-null    float64
16   16     1720 non-null   float64
17   17     1839 non-null   float64
18   18     1840 non-null   float64
19   19     1720 non-null   float64
20   20     1720 non-null   float64
21   21     1720 non-null   float64
dtypes: float64(22)
memory usage: 316.4 KB

```

### Kopf und Daten zusammenführen

Die Spalten mit dem Index 13 (Spalte N) und 15 (Spalte P) sind leer, diese werden aus dem Kopf und den Daten entfernt. Alle Spalten werden als Fließkommazahl erkannt. Das bedeutet, die Prozentzeichen in den Spalten mit den Indizes 16 und 19 bis 21 (Q, T:V) wurden durch eine Division durch 100 verarbeitet. Mit Ausnahme der Spalte Date wurde somit alle Datentypen korrekt erkannt.

```

# # leere Spalten entfernen
shiller_column_labels = shiller_column_labels.drop(labels = [13, 15])
shiller_data = shiller_data.drop(labels = [13, 15], axis = 'columns')

# Spaltennamen eintragen
shiller_data.columns = shiller_column_labels
shiller_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1840 entries, 0 to 1839
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
 ---  --  
 0   Date            1840 non-null   float64 
 1   S&PComp.P       1840 non-null   float64 
 2   DividendD       1839 non-null   float64 
 3   EarningsE       1836 non-null   float64 

```

```

4   ConsumerPriceIndexCPI           1840 non-null float64
5   DateFraction                   1840 non-null float64
6   LongInterestRateGS10          1840 non-null float64
7   RealPrice                      1840 non-null float64
8   RealDividend                   1839 non-null float64
9   RealTotalReturnPrice           1840 non-null float64
10  RealEarnings                   1836 non-null float64
11  RealTRScaledEarnings           1836 non-null float64
12  CAPE                          1720 non-null float64
13  TRCAPE                         1720 non-null float64
14  ExcessCAPEYield                1720 non-null float64
15  MonthlyTotalBondReturns        1839 non-null float64
16  RealTotalBondReturns           1840 non-null float64
17  10YearAnnualizedStockRealReturn 1720 non-null float64
18  10YearAnnualizedBondsRealReturn 1720 non-null float64
19  Real10YearExcessAnnualizedReturns 1720 non-null float64
dtypes: float64(20)
memory usage: 287.6 KB

```

### Datumsformat korrigieren

Im nächsten Schritt wird das Datumsformat korrigiert. Die Spalte Date enthält Zeichenketten im Format ‘YYYY,MM’. Eine Ausnahme ist der Monat Oktober, der im Format ‘YYYY.M’ kodiert ist. In der Ausgabe mit `print()` ist dies nicht zu sehen, da die Darstellung von 2 Dezimalstellen mit dem Befehl `pd.set_option("display.precision", 2)` eingestellt wurde. Die unterschiedliche Länge der Zeichketten kann mit dem Befehl `shiller_data.loc[0:12, 'Date'].astype('str').str.len()` verdeutlicht werden.

```

print(shiller_data.loc[0:12, 'Date'], "\n")

try:
    pd.to_datetime(shiller_data.loc[0:12, 'Date'], format = "%Y.%m")
except ValueError as error:
    print(error, "\n")
else:
    pd.to_datetime(shiller_data.loc[0:12, 'Date'], format = "%Y.%m")

print(shiller_data.loc[0:12, 'Date'].astype('str').str.len(), "\n")

0      1871.01
1      1871.02

```

```
2    1871.03
3    1871.04
4    1871.05
5    1871.06
6    1871.07
7    1871.08
8    1871.09
9    1871.10
10   1871.11
11   1871.12
12   1872.01
Name: Date, dtype: float64
```

```
time data "1871" doesn't match format "%Y.%m", at position 0. You might want to tr
  - passing `format` if your strings have a consistent format;
  - passing `format='ISO8601'` if your strings are all ISO8601 but not necessarily
    in the standard format;
  - passing `format='mixed'`, and the format will be inferred for each element in
```

```
0    7
1    7
2    7
3    7
4    7
5    7
6    7
7    7
8    7
9    6
10   7
11   7
12   7
Name: Date, dtype: int64
```

Der Datentyp der Spalte ‘Date’ wird als String deklariert. Mit einer Maske werden die Zeichenketten mit Länge 6 bestimmt. An die Zeichenketten mit der Länge 6 wird eine 0 angehängt, um das Datumsformat anzugeleichen. Anschließend wird die Spalte mit der Funktion `pd.to_datetime(format = "%Y.%m")` als datetime eingelesen.

```

# Datentyp Spalte Date als String setzen
# führt zu einer Fehlermeldung:
shiller_data['Date'] = shiller_data['Date'].astype('str');

maske = shiller_data['Date'].str.len() == 6
shiller_data.loc[maske, 'Date'] = shiller_data.loc[maske, 'Date'] + '0'

# Ausgabe Länge Zeichenkette
print(shiller_data.loc[0:12, 'Date'].str.len())
print(shiller_data.loc[0:12, 'Date'])

# Datentyp Spalte Date als datetime setzen
shiller_data['Date'] = pd.to_datetime(shiller_data['Date'], format = "%Y.%m");
shiller_data.info()

0      7
1      7
2      7
3      7
4      7
5      7
6      7
7      7
8      7
9      7
10     7
11     7
12     7
Name: Date, dtype: int64
0    1871.01
1    1871.02
2    1871.03
3    1871.04
4    1871.05
5    1871.06
6    1871.07
7    1871.08
8    1871.09
9    1871.10
10   1871.11
11   1871.12
12   1872.01

```

```

Name: Date, dtype: object
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1840 entries, 0 to 1839
Data columns (total 20 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   Date             1840 non-null   datetime64[ns]
 1   S&PComp.P        1840 non-null   float64 
 2   DividendD        1839 non-null   float64 
 3   EarningsE        1836 non-null   float64 
 4   ConsumerPriceIndexCPI 1840 non-null   float64 
 5   DateFraction     1840 non-null   float64 
 6   LongInterestRateGS10 1840 non-null   float64 
 7   RealPrice         1840 non-null   float64 
 8   RealDividend      1839 non-null   float64 
 9   RealTotalReturnPrice 1840 non-null   float64 
 10  RealEarnings       1836 non-null   float64 
 11  RealTRScaledEarnings 1836 non-null   float64 
 12  CAPE              1720 non-null   float64 
 13  TRCAPE             1720 non-null   float64 
 14  ExcessCAPEYield    1720 non-null   float64 
 15  MonthlyTotalBondReturns 1839 non-null   float64 
 16  RealTotalBondReturns 1840 non-null   float64 
 17  10YearAnnualizedStockRealReturn 1720 non-null   float64 
 18  10YearAnnualizedBondsRealReturn 1720 non-null   float64 
 19  Real10YearExcessAnnualizedReturns 1720 non-null   float64 
dtypes: datetime64[ns](1), float64(19)
memory usage: 287.6 KB

```

## 54 Zugriff auf mehrere lokale Dateien: Modul glob

glob

```
import glob
ordnerpfad = '01-daten/glob'

pfadliste = glob.glob(pathname = '*', root_dir = ordnerpfad, recursive = False)
print(pfadliste)
```

```
pfadliste = glob.glob(pathname = '**', root_dir = ordnerpfad, recursive = True)
print(pfadliste)
```

## Dokumentation des Moduls glob

💡 Tip ??: Tipp und Musterlösung

Tipp: Die gesuchten Dateien beginnen mit dem Buchstaben ‘U’ und enden mit der Dateiendung ‘.csv’. Wie Sie Dateien aus einem Ordner und aus einem Unterordner auslesen, sehen Sie in der verlinkten Dokumentation unter Examples.

### Tip 54.1: Musterlösung glob

```
pfadliste = glob.glob(pathname = '**/U*.csv', root_dir = ordnerpfad, recursive = True)
print(pfadliste)
```

```
['Unfallorte2020_LinRef.csv', 'Unfallorte2022_LinRef.csv', 'Unterordner glob/Unfallort
```

```
list_of_files = []
for pfade in pfadliste:
    zwischenspeicher = pd.read_csv(filepath_or_buffer = ordnerpfad + '/' + pfade,
                                    delimiter = ';', nrows = 3, usecols = [0, 1, 2])
    list_of_files.append(zwischenspeicher)
    print(pfade, "\n", zwischenspeicher, "\n")
```

```
unfalldaten2020 = list_of_files[0]
unfalldaten2021 = list_of_files[2]
unfalldaten2022 = list_of_files[1]
unfalldaten2023 = list_of_files[3]

print(unfalldaten2020, "\n")
print(unfalldaten2020.dtypes)
```

```
# Spalte OID_ / OBJECTID vereinheitlichen
unfalldaten = pd.concat([unfalldaten2020.iloc[:, 1:3], unfalldaten2022.iloc[:, 1:3], unfalldaten2018.iloc[:, 1:3]])
unfalldaten.insert(loc = 0, column = 'OBJECTID', value = pd.Series(range(1, unfalldaten.shape[0])))
unfalldaten = unfalldaten.astype('uint64')

unfalldaten
```

## 54.1 Übungen Modul glob

### 54.1.0.1 Leicht: US State Facts and Figures

- 
- 
- 
- 

[rdocumentation.org](https://rdocumentation.org)

- 
-

- 
- 

#### 💡 Tip ??: Musterlösung US State Facts and Figures

Mit dem Modul `glob` werden die Pfade der im Ordner liegenden Dateien in einer Liste gespeichert. In einer Schleife wird mit der Funktion `open()` jede Datei als Dateiobjekt geöffnet, aus dem der Dateiname und der Dateiinhalt ausgelesen werden, um diese zu betrachten.

```
import os
ordnerpfad = "01-daten/glob leicht"
pfadliste = glob.glob(pathname = '*', root_dir = ordnerpfad, recursive = False)

list_of_files = []
names_of_files = []
for pfad in pfadliste:

    # Dateiobjekt öffnen
    zwischenspeicher = open(ordnerpfad + '/' + pfad, 'r')

    # Dateinamen extrahieren
    name = os.path.basename(zwischenspeicher.name)
    names_of_files.append(name)

    # Datei auslesen
    datei = zwischenspeicher.read()
    list_of_files.append(datei)

    # Ausgabe
    print(name, "Encoding:", zwischenspeicher.encoding)
    print(datei[0:40], "\n")

    # Dateiobjekt schließen
    zwischenspeicher.close()

print(f"Die Dateien heißen:\n{names_of_files}")

state region.csv Encoding: UTF-8
```

```
"x"  
"1" "South"  
"2" "West"  
"3" "West"  
"4"  
  
state name.csv Encoding: UTF-8  
"x"  
"1" "Alabama"  
"2" "Alaska"  
"3" "Ariz  
  
state area.csv Encoding: UTF-8  
"x"  
"1" 51609  
"2" 589757  
"3" 113909  
"4"  
  
state abb.csv Encoding: UTF-8  
"x"  
"1" "AL"  
"2" "AK"  
"3" "AZ"  
"4" "AR"
```

Die Dateien heißen:

```
['state region.csv', 'state name.csv', 'state area.csv', 'state abb.csv']
```

Als nächstes werden die Daten in einem Datensatz ‘states’ gespeichert. Als Spaltennamen bieten sich die Dateinamen ohne Dateiendung an.

```

# Spaltennamen vorbereiten
i = 0
for name in names_of_files:
    names_of_files[i] = name[0:-4]
    i+= 1

# DataFrame erstellen
states = pd.DataFrame()
i = 0
list_of_files = []
for pfade in pfadliste:
    zwischenspeicher = open(ordnerpfad + '/' + pfade, 'r')
    states[i] = pd.read_csv(filepath_or_buffer = zwischenspeicher, sep = ' ', usecols = [1])
    zwischenspeicher.close()
    i+= 1

# Spaltennamen eintragen
states.columns = names_of_files

print(f"Es gibt die Regionen:\n{states['state region'].unique()}\n")

states.head()

```

Es gibt die Regionen:  
['South' 'West' 'Northeast' 'North Central']

	state region	state name	state area	state abb
0	South	Alabama	51609	AL
1	West	Alaska	589757	AK
2	West	Arizona	113909	AZ
3	South	Arkansas	53104	AR
4	West	California	158693	CA

Im nächsten Schritt können die Anzahl der Staaten und die Fläche je Region bestimmt werden.

```
print(f"Die Anzahl der Staaten je Region beträgt:\n{states.groupby('state region')['state name'].count()}")
print(f"Die Fläche der Regionen beträgt:\n{states.groupby('state region')['state area'].apply(lambda x: x.sum())}
```

Die Anzahl der Staaten je Region beträgt:

```
state region
North Central      12
Northeast          9
South               16
West                13
Name: state name, dtype: int64
```

Die Fläche der Regionen beträgt:

```
state region
North Central      765530
Northeast          169353
South               899556
West                1783960
Name: state area, dtype: int64
```

```
/var/folders/p_ks3trxjx0jd839_g4g0vm4nc0000gn/T/ipykernel_80936/3346728540.py:2: FutureWarning:
print(f"Die Fläche der Regionen beträgt:\n{states.groupby('state region')['state area'].apply(lambda x: x.sum())}")
```

#### 54.1.0.2 Schwer: DSB Unfallatlas

### 💡 Tip ??: Musterlösung Unfallatlas

Das Erhebungsjahr ist eine Variable im Datensatz. Dennoch wird gezeigt, wie dieses aus dem Dateinamen ausgelesen werden kann.

Zuerst werden der Inhalt des Ordners und anschließend die Dateipfade ausgelesen.

```
# Inhalt Ordner auslesen
ordnerpfad = '01-daten/glob schwer'
pfadliste = glob.glob(pathname = '**', root_dir = ordnerpfad, recursive = True)
print(f"Im Suchpfad liegen {len(pfadliste)} Elemente. Die Bezeichnungen lauten:\n{pfadliste}\n")

# Dateipfade auslesen
pfadliste = glob.glob(pathname = '**/U*.csv', root_dir = ordnerpfad, recursive = True)
print(f"Im Suchpfad liegen {len(pfadliste)} CSV-Dateien. Die Dateinamen lauten:\n{pfadliste}\n")
```

Im Suchpfad liegen 6 Elemente. Die Bezeichnungen lauten:

```
['Unfallorte2020_LinRef.csv', 'Unfallorte2022_LinRef.csv', 'DSB_Unfallatlas Metadaten.pdf',
```

Im Suchpfad liegen 4 CSV-Dateien. Die Dateinamen lauten:

```
['Unfallorte2020_LinRef.csv', 'Unfallorte2022_LinRef.csv', 'Unterordner glob/Unfallorte2022_LinRef.csv']
```

Als nächstes werden die Jahreszahlen aus den Pfadnamen extrahiert sowie die Spaltennamen ausgelesen.

Um die Jahreszahlen aus den Dateinamen zu extrahieren, können die Zeichenketten manuell beschnitten werden (Die Jahreszahl steht an den Indexpositionen 10 bis exklusiv 14). Alternativ kann ein regulärer Ausdruck verwendet werden. Reguläre Ausdrücke werden mit dem Modul `re` verarbeitet. Die Funktion `re.search(string = x, pattern = '[0-9]+')` sucht im Objekt `x` nach dem im Argument `pattern` spezifizierten Ausdruck, hier nach einer Zahl von 0 bis 9 [0-9] und jeder darauf folgenden Zahl +. Die Funktion `re.search()` erzeugt ein Match-Objekt, dessen Inhalt mit der Methode `Match.group()` ausgegeben werden kann, die einen string zurückgibt. (siehe [Dokumentation Modul re](#)) Um die Spaltennamen auszulesen, wird die Funktion `pd.read_csv()` mit den Argumenten `nrows = 1, header = 0` angewiesen, die erste Zeile auszulesen. Anschließend können die Spaltennamen visuell abgeglichen werden - dies ist im Reiter DataFrame der Spaltennamen umgesetzt. Für größere Datensätze ist dies aber nicht mehr praktikabel. Eine algorithmische Lösung ist im Reiter Spaltennamen finden umgesetzt. Dazu wird jede Spalte des DataFrames gegen alle anderen Spalten mit der Methode `pd.Series.isin(df)` abgeglichen.

## 54.2 Jahreszahlen und Spaltennamen auslesen

```
# Die Jahreszahlen auslesen
import re

jahreszahlen = []
for pfade in pfadliste:
    zwischenspeicher = re.search(string = pfade, pattern = '[0-9]+').group()
    jahreszahlen.append(int(zwischenspeicher))

print(f"Aus den Dateinamen extrahierte Jahreszahlen:\t{jahreszahlen}\n")

# Spaltennamen durch Einlesen der ersten Zeile extrahieren, header = None
i = 0
list_of_columns = []
for pfade in pfadliste:

    zwischenspeicher = pd.read_csv(filepath_or_buffer = ordnerpfad + '/' + pfade,
                                    delimiter = ';', nrows = 1, header = None).transpose()

    list_of_columns.append(zwischenspeicher)
    i += 1

df_of_columns = pd.concat(list_of_columns, axis = 1, ignore_index = True)
df_of_columns.columns = jahreszahlen
```

Aus den Dateinamen extrahierte Jahreszahlen: [2020, 2022, 2021, 2023]

Die Spaltennamen werden im nächsten Reiter ausgegeben.

## 54.3 DataFrame der Spaltennamen

Mit der Ausgabe des DataFrames können die ungleichen Spaltennamen visuell identifiziert werden.

```
print(f"Ein DataFrame der Spaltennamen:\n{df_of_columns}")
```

Ein DataFrame der Spaltennamen:

	2020	2022	2021	2023
0	OBJECTID	OBJECTID	OBJECTID	OID_
1	UIDENTSTLAE	UIDENTSTLAE	UIDENTSTLAE	UIDENTSTLAE

2	ULAND	ULAND	ULAND	ULAND
3	UREGBEZ	UREGBEZ	UREGBEZ	UREGBEZ
4	UKREIS	UKREIS	UKREIS	UKREIS
5	UGEMEINDE	UGEMEINDE	UGEMEINDE	UGEMEINDE
6	UJAHR	UJAHR	UJAHR	UJAHR
7	UMONAT	UMONAT	UMONAT	UMONAT
8	USTUNDE	USTUNDE	USTUNDE	USTUNDE
9	UWOCHENTAG	UWOCHENTAG	UWOCHENTAG	UWOCHENTAG
10	UKATEGORIE	UKATEGORIE	UKATEGORIE	UKATEGORIE
11	UART	UART	UART	UART
12	UTYP1	UTYP1	UTYP1	UTYP1
13	ULICHTVERH	ULICHTVERH	ULICHTVERH	ULICHTVERH
14	IstRad	IstStrassenzustand	USTRZUSTAND	IstStrassenzustand
15	IstPKW	IstRad	IstRad	IstRad
16	IstFuss	IstPKW	IstPKW	IstPKW
17	IstKrad	IstFuss	IstFuss	IstFuss
18	IstGkfz	IstKrad	IstKrad	IstKrad
19	IstSonstige	IstGkfz	IstGkfz	IstGkfz
20	LINREFX	IstSonstige	IstSonstige	IstSonstige
21	LINREFY	LINREFX	LINREFX	LINREFX
22	XGCSWGS84	LINREFY	LINREFY	LINREFY
23	YGCSWGS84	XGCSWGS84	XGCSWGS84	XGCSWGS84
24	STRZUSTAND	YGCSWGS84	YGCSWGS84	YGCSWGS84
25	NaN	NaN	NaN	PLST

## 54.4 Spaltennamen finden

Der Abgleich der ungleichen Spaltennamen kann auch algorithmisch gelöst werden.

```

# Spaltennamen algorithmisch abgleichen
ungleiche_spaltennamen = pd.Series()

## vergleiche jede Spalte gegen alle anderen Spalten
for i in range(0, df_of_columns.shape[1]):

    aktuelle_spalte = df_of_columns.iloc[:, i]

    vergleichs_df = df_of_columns.drop(df_of_columns.columns[i], axis = 1)

    j = 0
    for vergleichsspalte in vergleichs_df: # vergleichsspalte ist der Spaltenname (als int) 2

        # print(vergleichsspalte)
        maske_fehlender_spaltennamen = aktuelle_spalte.isin(vergleichs_df.loc[:, vergleichsspalte])
        maske_fehlender_spaltennamen = np.invert(maske_fehlender_spaltennamen) # pandas hat kein

        # WENN Spaltennamen fehlen DANN
        if maske_fehlender_spaltennamen.sum() > 0:

            print(f"In den Unfalldaten {df_of_columns.columns[i]} treten die Spaltennamen {aktueller_spalte} auf")

            ungleiche_spaltennamen = pd.concat([ungleiche_spaltennamen, pd.Series(aktuelle_spalte)], ignore_index=True)

    j +=1

ungleiche_spaltennamen = ungleiche_spaltennamen.dropna().unique()

print(f"Folgende Spaltennamen müssen vereinheitlicht werden:\n{ungleiche_spaltennamen}")


In den Unfalldaten 2020 treten die Spaltennamen ['STRZUSTAND'] auf, die nicht in den Unfall

In den Unfalldaten 2020 treten die Spaltennamen ['STRZUSTAND'] auf, die nicht in den Unfall

In den Unfalldaten 2020 treten die Spaltennamen ['OBJECTID' 'STRZUSTAND' nan] auf, die nicht in

In den Unfalldaten 2022 treten die Spaltennamen ['IstStrassenzustand'] auf, die nicht in de

In den Unfalldaten 2022 treten die Spaltennamen ['IstStrassenzustand'] auf, die nicht in de

In den Unfalldaten 2022 treten die Spaltennamen ['OBJECTID' nan] auf, die nicht in den Unfa

```

In den Unfalldaten 2021 treten die Spaltennamen ['USTRZUSTAND'] auf, die nicht in den Unfallatlas Metadaten vorkommen.

In den Unfalldaten 2021 treten die Spaltennamen ['USTRZUSTAND'] auf, die nicht in den Unfallatlas Metadaten vorkommen.

In den Unfalldaten 2021 treten die Spaltennamen ['OBJECTID' 'USTRZUSTAND' nan] auf, die nicht in den Unfallatlas Metadaten vorkommen.

In den Unfalldaten 2023 treten die Spaltennamen ['OID\_' 'IstStrassenzustand' 'PLST'] auf, die nicht in den Unfallatlas Metadaten vorkommen.

In den Unfalldaten 2023 treten die Spaltennamen ['OID\_' 'PLST'] auf, die nicht in den Unfallatlas Metadaten vorkommen.

In den Unfalldaten 2023 treten die Spaltennamen ['OID\_' 'IstStrassenzustand' 'PLST'] auf, die nicht in den Unfallatlas Metadaten vorkommen.

Folgende Spaltennamen müssen vereinheitlicht werden:  
['STRZUSTAND' 'OBJECTID' 'IstStrassenzustand' 'USTRZUSTAND' 'OID\_' 'PLST']

Die Spaltennamen ‘STRZUSTAND’, ‘IstStrassenzustand’ und ‘USTRZUSTAND’ sowie ‘OBJECTID’, und ‘OID\_’ müssen beim Einlesen der Daten vereinheitlicht werden. Die Spalte ‘PLST’ wurde nur im Jahr 2023 erhoben und kann entfallen (siehe Unfallatlas Metadaten). Dafür wird der Methode `pd.DataFrame.drop()` das Argument `errors = 'ignore'` übergeben, da in einem Datensatz nicht vorhandene Spalten andernfalls zu einer Fehlermeldung führen.

Die Spalte ‘UIDENTSTLAE’ enthält sehr große Ganzzahlen, weshalb diese als String eingelesen wird.

```

## Dateien einlesen
unfallatlas = pd.DataFrame()

for pfad in pfadliste:

    ### Datei einlesen
    zwischenspeicher = pd.read_csv(filepath_or_buffer = ordnerpfad + '/' + pfad,
    delimiter = ';', dtype = {'UIDENTSTLAE': 'string'})

    ### Spaltennamen vereinheitlichen
    zwischenspeicher.rename(columns = {"IstStrassenzustand": "STRZUSTAND", "USTRZUSTAND": "STZUSTAND", "PLST": "PLSTLAE"}, inplace = True, errors = 'ignore')

    unfallatlas = pd.concat([unfallatlas, zwischenspeicher])

# Ausgabe
print(unfallatlas.info(), "\n")
unfallatlas.head()

<class 'pandas.core.frame.DataFrame'>
Index: 996742 entries, 0 to 269047
Data columns (total 25 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   OBJECTID        996742 non-null   int64  
 1   UIDENTSTLAE    996742 non-null   string 
 2   ULAND           996742 non-null   int64  
 3   UREGBEZ         996742 non-null   int64  
 4   UKREIS          996742 non-null   int64  
 5   UGEMEINDE       996742 non-null   int64  
 6   UJAHR           996742 non-null   int64  
 7   UMONAT          996742 non-null   int64  
 8   USTUNDE         996742 non-null   int64  
 9   UWOCHENTAG      996742 non-null   int64  
 10  UKATEGORIE     996742 non-null   int64  
 11  UART            996742 non-null   int64  
 12  UTYP1           996742 non-null   int64  
 13  ULICHTVERH      996742 non-null   int64  
 14  IstRad          996742 non-null   int64  
 15  IstPKW          996742 non-null   int64  
 16  IstFuss          996742 non-null   int64  
 17  IstKrad          996742 non-null   int64

```

```

18  IstGkfz      996742 non-null  int64
19  IstSonstige   996742 non-null  int64
20  LINREFX      996742 non-null  object
21  LINREFY      996742 non-null  object
22  XGCSWGS84    996742 non-null  object
23  YGCSWGS84    996742 non-null  object
24  STRZUSTAND   996742 non-null  int64
dtypes: int64(20), object(4), string(1)
memory usage: 197.7+ MB
None

```

	OBJECTID	UIDENTSTLAE	ULAND	UREGBEZ	UKREIS	UGEMEINDE	UJAHR	UMONA
0	1	12200116471201851100	12	0	68	468	2020	1
1	2	12200106642131830700	12	0	61	112	2020	1
2	3	12200109522101836720	12	0	67	144	2020	1
3	4	12200131722601877310	12	0	69	76	2020	1
4	5	12200124632322878920	12	0	62	224	2020	1

Abschließend kann der Datensatz aufgeräumt werden:

- der Datensatz wird nach dem Jahr sortiert (wichtig: als erstes ausführen),
- der Index wird zurückgesetzt (wichtig: als zweites),
- die Spalte ‘OBJECTID’ wird neu erstellt, um die beim Einlesen erzeugten Duplikate zu entfernen und

```
# Datensatz nach Jahr sortieren
unfallatlas.sort_values(by = 'UJAHR', inplace = True)

# Index zurücksetzen
unfallatlas.reset_index(drop = True, inplace = True)

# Spalte 'OBJECTID' neu generieren
unfallatlas['OBJECTID'] = pd.Series(range(1, unfallatlas.shape[0] + 1))

print(unfallatlas.info(), "\n")
unfallatlas.head()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 996742 entries, 0 to 996741
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   OBJECTID          996742 non-null   int64  
 1   UIDENTSTLAE       996742 non-null   string  
 2   ULAND              996742 non-null   int64  
 3   UREGBEZ           996742 non-null   int64  
 4   UKREIS             996742 non-null   int64  
 5   UGEMEINDE          996742 non-null   int64  
 6   UJAHR              996742 non-null   int64  
 7   UMONAT             996742 non-null   int64  
 8   USTUNDE            996742 non-null   int64  
 9   UWOCHENTAG         996742 non-null   int64  
 10  UKATEGORIE         996742 non-null   int64  
 11  UART               996742 non-null   int64  
 12  UTYP1              996742 non-null   int64  
 13  ULICHTVERH         996742 non-null   int64  
 14  IstRad             996742 non-null   int64  
 15  IstPKW             996742 non-null   int64
```

```

16 IstFuss      996742 non-null  int64
17 IstKrad      996742 non-null  int64
18 IstGkfz      996742 non-null  int64
19 IstSonstige   996742 non-null  int64
20 LINREFX      996742 non-null  object
21 LINREFY      996742 non-null  object
22 XGCSWGS84    996742 non-null  object
23 YGCSWGS84    996742 non-null  object
24 STRZUSTAND   996742 non-null  int64
dtypes: int64(20), object(4), string(1)
memory usage: 190.1+ MB
None

```

	OBJECTID	UIDENTSTLAE	ULAND	UREGBEZ	UKREIS	UGEMEINDE	UJAHR	UMONA
0	1	12200116471201851100	12	0	68	468	2020	1
1	2	09200116005123027960	9	5	77	177	2020	1
2	3	09200119006203025660	9	6	72	114	2020	1
3	4	09200116004301008520	9	4	73	120	2020	1
4	5	09200119001515007880	9	1	83	128	2020	1

## 55 Maskierte Arrays

“**nan**, short for ‘not a number’ [...] was specifically designed to address the problem of missing values, but the reality is that different platforms behave differently, making life more difficult. On some platforms, the presence of **nan** slows calculations 10-100 times. For integer data, no **nan** value exists. [...] Those wishing to avoid potential headaches will be interested in an alternative solution which has a long history in NumPy’s predecessors – **masked arrays**. [...] Despite their additional memory requirement, masked arrays are faster than nans on many floating point units.” (SciPy.org Frequently Asked Questions, Zugriff via [waybackmachine](#))

```
# Fehlende Werte für Ganzzahlen sind in Pandas kein Problem
print("Eine pd.Series vom dtype Int64 mit einem fehlenden Wert:")
print(pd.Series([0, 1, np.nan], dtype = 'Int64'), "\n")

# In NumPy ist das anders
try:
    np.array([0, 1, np.nan], dtype = 'int')
except ValueError as error:
    print("Da np.nan vom Datentyp float ist, haben Integer-Arrays keinen fehlenden Wert.\n Die
else:
    np.array([0, 1, np.nan], dtype = 'int')

print(f"Die Ursache ist der Datentyp von nan: type(np.nan)\n{type(np.nan)}\n")
```

1.

2.

3.

```
import numpy.ma as ma

maskiertes_array = ma.masked_array(data = np.array([1, 2, 3, 4]), mask = [0, 0, 1, 1])
print(f"maskiertes Array:\t{maskiertes_array}")

print(f"Daten:\t\t\t{maskiertes_array.data}")

print(f"Maske:\t\t\t{maskiertes_array.mask}")

print(f"Zugriff über Funktionen ma.getdata() und ma.getmask():\n{ma.getdata(maskiertes_array)}")

print(f"gefülltes Array:\t\t\t{maskiertes_array.filled()}")

print(f"Das maskierte Array bleibt unverändert: {maskiertes_array}")
```

```
print(np.ma.default_fill_value(1))
print(np.ma.default_fill_value('1'))
```

## NumPy Dokumentation

```
# Anlegen eines maskierten Arrays vom Datentyp Integer mit einem ganzzahligen Füllwert
maskiertes_array = ma.masked_array(data = np.arange(0, 4), mask = [0, 0, 1, 1], fill_value = 1)

print(f"Gefülltes Array vom Datentyp integer mit ganzzahligem Füllwert:\n{maskiertes_array.fill_value}")

# Ändern des Füllwerts mit Datentyp float
maskiertes_array.fill_value = 1.5
print(f"Wird einem maskierten Array vom Datentyp integer ein Füllwert vom Datentyp float übergeben")
```

```

print(f"Wird einem maskierten Array vom Datentyp integer ein Füllwert vom Datentyp string übe

try:
    maskiertes_array.fill_value = '1.5'
    print(maskiertes_array.filled())
except TypeError as error:
    print(error)
else:
    maskiertes_array.fill_value = '1.5'
    print(maskiertes_array.filled())

# Füllwert None
maskiertes_array.fill_value = None
print(f"\nDer Füllwert None bewirkt den Rückgriff auf einen datentypabhängigen Standardwert:")

```

## 55.1 Maskierte Arrays erzeugen

[Dokumentation](#)

```

a = np.array([1, 10, 100, 1000])

print(f"Die Methode ma.asanyarray(array, dtype = ) legt den Datentyp des maskierten Arrays f
      f"ma.asanyarray(a, dtype = 'float64'):\n"
      f"{{ma.asanyarray(a, dtype = 'float64')}}\n")

```

```
print(f"Die Methode ma.masked_equal(x = array, value) maskiert alle Werte value.\n"
      f"ma.masked_equal(a, 100):\n"
      f"\t{ma.masked_equal(a, 100)}\n")

print(f"Die Methode ma.masked_greater(x = array, value) maskiert alle Werte größer als value\n"
      f"ma.masked_greater(a, 100):\n"
      f"\t{ma.masked_greater(a, 100)}\n")

print(f"Die Methode ma.masked_inside(x = array, v1, v2) maskiert alle Werte im Intervall v1 bis v2\n"
      f"ma.masked_inside(a, 5, 100):\n"
      f"\t{ma.masked_inside(a, 5, 100)}\n")

print(f"Die Methode ma.masked_where(condition, a = array) maskiert alle Werte, für die die Bedingung\n"
      f"ma.masked_where(a % 2 != 0, a):\n"
      f"\t{ma.masked_where(a % 2 != 0, a)}\n")
```

```
dateipfad = '01-daten/TC01_double_hyphen.csv'  
daten_double_hyphen = np.genfromtxt(dateipfad, missing_values = '--', usemask = True)  
daten_double_hyphen
```

```
maskiertes_array = ma.masked_array([1, 2, 3, 4])  
  
maskiertes_array[1] = ma.masked  
print(maskiertes_array)  
  
maskiertes_array[1:3] = ma.masked  
print(maskiertes_array)
```

## 55.2 Übung maskierte Arrays erzeugen

- a.
- b.
- c.
- d.

💡 Tip ??: Musterlösung maskierte Arrays erzeugen

```
# Array erzeugen
a = np.linspace(1, 1000, 18, dtype = 'int')

# Maske im Intervall 250-750
my_ma = ma.masked_inside(x = a, v1 = 250, v2 = 750)
print("Maske im Intervall 250-750:")
print(my_ma, "Anzahl maskierter Werte:", my_ma.mask.sum(), "\n")

# jeder 2. Wert maskiert
my_ma = ma.masked_array(a)
my_ma[::2] = ma.masked
print("Jeder 2. Wert maskiert:")
print(my_ma, "Summe nicht maskierter Werte:", my_ma.sum(), "\n")

# jeder gerade Wert maskiert
my_ma = ma.masked_where(a % 2 == 0, a)
print("Jeder gerade Wert maskiert:")
print(my_ma, "\n")

# jeder mindestens dreistellige Wert maskiert
my_ma = ma.masked_where(a / 100 >= 1, a)
print("Jeder mindestens dreistellige Wert maskiert:")
print(my_ma)

## eine Alternative
my_mask = [len(str(i)) >= 3 for i in a]
print(f"ma.masked_array(data = a, mask = my_mask")
```

Maske im Intervall 250-750:  
[1 59 118 177 236 -- -- -- -- -- -- -- -- 764 823 882 941 1000] Anzahl maskierter Werte: 8

Jeder 2. Wert maskiert:  
[-- 59 -- 177 -- 294 -- 412 -- 529 -- 647 -- 764 -- 882 -- 1000] Summe nicht maskierter Werte: 4464

Jeder gerade Wert maskiert:  
[1 59 -- 177 -- -- 353 -- 471 529 -- 647 -- -- 823 -- 941 --]

Jeder mindestens dreistellige Wert maskiert:  
[1 59 -- -- -- -- -- -- -- -- -- -- -- -- -- --]

```
ma.masked_array(data = a, mask = my_mask
```

### 55.3 unmasking, soft und hard masks

```
unmask_me = my_ma.copy()
unmask_me.mask = ma.nomask
print(f"Die Maske kann auf ma.nomask oder False gesetzt werden.\n"
      f"unmask_me.mask = ma.nomask: {unmask_me}\n")

unmask_me = my_ma.copy()
unmask_me.mask = False
print(f"unmask_me.mask = False: {unmask_me}\n")

unmask_me = my_ma.copy()
unmask_me = np.linspace(1, 1000, 18, dtype = 'int')
print(f"Die Maskierung wird auch durch eine Wertzuweisung aufgehoben.\nDas Objekt wird neu angesetzt.\n"
      f"unmask_me = np.linspace(1, 1000, 18, dtype = 'int'):\n"
      f"{unmask_me}\n")
```

```
unmask_me = my_ma.copy()
unmask_me[6:10].mask = ma.nomask
print(f"Die Maske kann auf ma.nomask oder False gesetzt werden.\n"
      f"unmask_me[6:10].mask = ma.nomask: {unmask_me}\n")

unmask_me = my_ma.copy()
unmask_me[6:10].mask = False
print(f"unmask_me[6:10].mask = False: {unmask_me}\n")
```

```

unmask_me = my_ma.copy()
unmask_me[6:10] = np.linspace(1, 1000, 4, dtype = 'int')
print(f"Die Maskierung wird auch durch eine Wertzuweisung aufgehoben.")
print(f"unmask_me[6:10] = np.linspace(1, 1000, 4, dtype = 'int'):\n"
      f"{unmask_me}\n")

```

### 55.3.0.1 soft und hard masks

```

unmask_me = my_ma.copy()
unmask_me.harden_mask()
unmask_me.mask = ma.nomask
print(f"Die hard mask wird auf ma.nomask gesetzt.\n"
      f"unmask_me.mask = ma.nomask: {unmask_me}\n")

unmask_me.mask = False
print(f"unmask_me.mask = False: {unmask_me}\n")

unmask_me = np.linspace(1, 1000, 18, dtype = 'int')
print(f"Die hard mask wird dennoch durch eine Wertzuweisung aufgehoben.\nDas Objekt wird neu"
      print(f"unmask_me = np.linspace(1, 1000, 18, dtype = 'int'):\n"
            f"{unmask_me}\n")

```

```
unmask_me = my_ma.copy()
unmask_me.harden_mask()
unmask_me[6:10].mask = ma.nomask
print(f"Die hard mask wird auf ma.nomask gesetzt.\n"
      f"unmask_me.mask[6:10] = ma.nomask: {unmask_me}\n")

unmask_me[6:10].mask = False
print(f"unmask_me[6:10].mask = False: {unmask_me}\n")

unmask_me[6:10] = np.linspace(1, 1000, 4, dtype = 'int')
print(f"Die hard mask wird nicht (!) durch eine Wertzuweisung im Indexbereich aufgehoben.")
print(f"unmask_me[6:10] = np.linspace(1, 1000, 4, dtype = 'int'): {unmask_me}\n"
      f"\n")
```

```
unmask_me[0:2] = ma.masked
print(f"Die Maskierungen zusätzlicher Elemente ist mit einer hard mask möglich.\n"
      f"\n")
```

```
unmask_me.soften_mask()
unmask_me.mask = ma.nomask
print(f"{unmask_me}")
```

NumPy Dokumentation

## 55.4 Operationen mit maskierten Arrays

<https://numpy.org/doc/stable/reference/maskedarray.generic.html#operations-on-masked-arrays>

<https://numpy.org/doc/2.1/reference/routines.ma.html>

```
# Array erzeugen
a = np.linspace(1, 1000, 18, dtype = 'int')
maskiertes_array = ma.masked_array(a)
maskiertes_array[::2] = ma.masked

# Ausgewählte Operationen
print(f"Summe maskiertes_array: {maskiertes_array.sum()}\nSumme maskiertes_array.data: {maskiertes_array.sum().data}")
print(f"maskiertes_array > 222:\n{maskiertes_array > 222}")
```

```
print("NumPy ist bei Operationen mit np.nan nicht konsistent.")
print(f"np.nan ** 0:{np.nan ** 0}")
print(f"1 ** np.nan:{1 ** np.nan}")
print(f"np.nan == np.nan:{np.nan == np.nan}\n")

print("Maskierte Arrays verhalten sich dagegen konsistent.")
print(f"maskiertes_array[0] ** 0:{maskiertes_array[0] ** 0}")
print(f"1 ** maskiertes_array[0]:{1 ** maskiertes_array[0]}")
print(f"maskiertes_array[0] == maskiertes_array[2]:{maskiertes_array[0] == maskiertes_array[2]}")
print(f"maskiertes_array[0] == np.nan:{maskiertes_array[0] == np.nan}")
```

[NumPy Dokumentation](#)

```
ma.log([-1, 0, 1, 2])
```

```
print(a + maskiertes_array)
print(np.logical_or(a, maskiertes_array))
```

```
# Ausgewählte Operationen
print(len(maskiertes_array))
print(ma.unique(maskiertes_array))
```

### ⚠ Warning ??: Warnhinweis maskierte Arrays

Die Dokumentation warnt, dass maskierte Werte nicht zuverlässig von Operationen ausgenommen sind. Welche Operationen dies betrifft, ist nicht angegeben.

“Arithmetic and comparison operations are supported by masked arrays. As much as possible, invalid entries of a masked array are not processed, meaning that the corresponding `data` entries *should* be the same before and after the operation.

#### Warning

We need to stress that this behavior may not be systematic, that masked data may be affected by the operation in some cases and therefore users should not rely on this data remaining unchanged.”

<https://numpy.org/doc/stable/reference/maskedarray.generic.html#operations-on-masked-arrays>

## 55.5 Übungen Operationen mit maskierten Arrays

1.

2.

3.

4.

5.

Sieb des

Eratosthenes

💡 Tip ??: Musterlösung

```
# 1. NumPy-Array erstellen und maskieren
maskiertes_array = np.arange(1, 100 + 1).reshape(10, 10)
maskiertes_array = ma.masked_where(condition = (maskiertes_array % 2 != 0) & (maskiertes_ar
```

```
[[1 2 -- 4 5 6 7 8 -- 10]
 [11 12 13 14 -- 16 17 18 19 20]
 [-- 22 23 24 25 26 -- 28 29 30]
 [31 32 -- 34 35 36 37 38 -- 40]
 [41 42 43 44 -- 46 47 48 49 50]
 [-- 52 53 54 55 56 -- 58 59 60]
 [61 62 -- 64 65 66 67 68 -- 70]
 [71 72 73 74 -- 76 77 78 79 80]
 [-- 82 83 84 85 86 -- 88 89 90]
 [91 92 -- 94 95 96 97 98 -- 100]]
```

```
# 2. Summen bilden
print(f"Summe der maskierten Werte: {maskiertes_array.data[maskiertes_array.mask].sum()}\tS
```

```
Summe der maskierten Werte: 867 Summe der unmaskierten Werte: 4183
```

```
# 3. Maske zurücksetzen
maskiertes_array.mask = False
print(maskiertes_array, "\n")
```

```
[[1 2 3 4 5 6 7 8 9 10]
 [11 12 13 14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27 28 29 30]
 [31 32 33 34 35 36 37 38 39 40]
 [41 42 43 44 45 46 47 48 49 50]
 [51 52 53 54 55 56 57 58 59 60]
 [61 62 63 64 65 66 67 68 69 70]
 [71 72 73 74 75 76 77 78 79 80]
 [81 82 83 84 85 86 87 88 89 90]
 [91 92 93 94 95 96 97 98 99 100]]
```

```

# 4. Primzahlsieb
## 1
i = 1
bedingung1 = maskiertes_array == i
sieg = ma.masked_where(bedingung1, maskiertes_array)
print(f"i = {i}\n{sieg}\n")

## 2
i = 2
bedingung2 = (maskiertes_array > i) & (maskiertes_array % i == 0)
sieg = ma.masked_where(bedingung1 | bedingung2, maskiertes_array)
print(f"zusätzlich Vielfache von {i}\n{sieg}\n")

## 3
i = 3
bedingung3 = (maskiertes_array > i) & (maskiertes_array % i == 0)
sieg = ma.masked_where(bedingung1 | bedingung2 | bedingung3, maskiertes_array)
print(f"zusätzlich Vielfache von {i}\n{sieg}\n")

## 5
i = 5
bedingung5 = (maskiertes_array > i) & (maskiertes_array % i == 0)
sieg = ma.masked_where(bedingung1 | bedingung2 | bedingung3 | bedingung5, maskiertes_array)
print(f"zusätzlich Vielfache von {i}\n{sieg}\n")

## 7
i = 7
bedingung7 = (maskiertes_array > i) & (maskiertes_array % i == 0)
sieg = ma.masked_where(bedingung1 | bedingung2 | bedingung3 | bedingung5 | bedingung7, maskiertes_array)
print(f"zusätzlich Vielfache von {i}\n{sieg}\n")

i = 1
[[-- 2 3 4 5 6 7 8 9 10]
 [11 12 13 14 15 16 17 18 19 20]
 [21 22 23 24 25 26 27 28 29 30]
 [31 32 33 34 35 36 37 38 39 40]
 [41 42 43 44 45 46 47 48 49 50]
 [51 52 53 54 55 56 57 58 59 60]
 [61 62 63 64 65 66 67 68 69 70]
 [71 72 73 74 75 76 77 78 79 80]
 [81 82 83 84 85 86 87 88 89 90]

```

```
[91 92 93 94 95 96 97 98 99 100]]
```

zusätzlich Vielfache von 2

```
[[-- 2 3 -- 5 -- 7 -- 9 --]
 [11 -- 13 -- 15 -- 17 -- 19 --]
 [21 -- 23 -- 25 -- 27 -- 29 --]
 [31 -- 33 -- 35 -- 37 -- 39 --]
 [41 -- 43 -- 45 -- 47 -- 49 --]
 [51 -- 53 -- 55 -- 57 -- 59 --]
 [61 -- 63 -- 65 -- 67 -- 69 --]
 [71 -- 73 -- 75 -- 77 -- 79 --]
 [81 -- 83 -- 85 -- 87 -- 89 --]
 [91 -- 93 -- 95 -- 97 -- 99 --]]
```

zusätzlich Vielfache von 3

```
[[-- 2 3 -- 5 -- 7 -- -- --]
 [11 -- 13 -- -- -- 17 -- 19 --]
 [-- -- 23 -- 25 -- -- -- 29 --]
 [31 -- -- -- 35 -- 37 -- -- --]
 [41 -- 43 -- -- -- 47 -- 49 --]
 [-- -- 53 -- 55 -- -- -- 59 --]
 [61 -- -- -- 65 -- 67 -- -- --]
 [71 -- 73 -- -- -- 77 -- 79 --]
 [-- -- 83 -- 85 -- -- -- 89 --]
 [91 -- -- -- 95 -- 97 -- -- --]]
```

zusätzlich Vielfache von 5

```
[[-- 2 3 -- 5 -- 7 -- -- --]
 [11 -- 13 -- -- -- 17 -- 19 --]
 [-- -- 23 -- -- -- -- 29 --]
 [31 -- -- -- -- 37 -- -- --]
 [41 -- 43 -- -- -- 47 -- 49 --]
 [-- -- 53 -- -- -- -- 59 --]
 [61 -- -- -- -- 67 -- -- --]
 [71 -- 73 -- -- -- 77 -- 79 --]
 [-- -- 83 -- -- -- -- 89 --]
 [91 -- -- -- -- 97 -- -- --]]
```

zusätzlich Vielfache von 7

```
[[-- 2 3 -- 5 -- 7 -- -- --]
 [11 -- 13 -- -- -- 17 -- 19 --]]
```

```
[-- -- 23 -- -- -- -- -- 29 --]
[31 -- -- -- -- 37 -- -- --]
[41 -- 43 -- -- -- 47 -- -- --]
[-- -- 53 -- -- -- -- 59 --]
[61 -- -- -- -- 67 -- -- --]
[71 -- 73 -- -- -- -- 79 --]
[-- -- 83 -- -- -- -- 89 --]
[-- -- -- -- -- 97 -- -- --]]
```

```
# 5. verschiedene Aufgaben
## Wie viele Primzahlen sind in dem Array?
print("Anzahl Primzahlen:", sieb.size - sieb.mask.sum())

## Welches ist die größte, welches die kleinste Primzahl?
print(sieb.max(), sieb.min())

## Primzahlen absteigend sortiert
### ma.sort() sortiert nur aufsteigend, deshalb:
### Übergabe des eindimensionalen arrays sieb.flatten() in umgekehrter Reihenfolge [::-1]
### anschließend umformen in 2D-Array
### endwith = False missing values are treated as smallest values
print(ma.sort(sieb.flatten(), endwith = False)[::-1].reshape(sieb.shape))
```

```
Anzahl Primzahlen: 25
97 2
[[97 89 83 79 73 71 67 61 59 53]
 [47 43 41 37 31 29 23 19 17 13]
 [11 7 5 3 2 -- -- -- -- --]
 [-- -- -- -- -- -- -- -- --]
 [-- -- -- -- -- -- -- -- --]
 [-- -- -- -- -- -- -- -- --]
 [-- -- -- -- -- -- -- -- --]
 [-- -- -- -- -- -- -- -- --]
 [-- -- -- -- -- -- -- -- --]
 [-- -- -- -- -- -- -- -- --]]
```

# 56 Wissenschaftliche Dateiformate

Key Facts

and Figures - CERN Data Centre

## 56.1 HDF5

- 
-

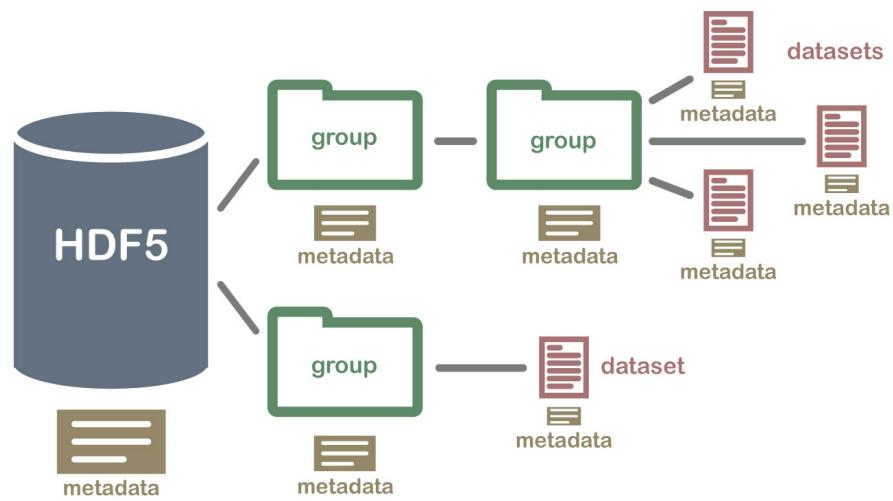


Figure 56.1: HDF5-Verzeichnisstruktur

<https://www.neonscience.org/resources/learning-hub/tutorials/about-hdf5>

PyTables

Thema auf stackoverflow Antwort von Kevin S unter CC-BY-SA 3.0

### **⚠ Warning ??: Datensatz**

In diesem Abschnitt wird ein Teil des Datensatzes IceBridge ATM L1B Elevation and Return Strength, Version 2 genutzt ([kostenlose Registrierung bei NASA Earth erforderlich](#)). Das [Handbuch ist hier verfügbar](#). Der Datensatz enthält flugzeugbasierte Lasermessungen für die Höhe des Eisschildes in der Arktis und Antarktis mittels des NASA Airborne Topographic Mapper (ATM). ATM hat eine Abtastrate von 3 bis 5 kHz. Jede Datei enthält die Daten von einigen Minuten Flugdauer. Das geografische Referenzsystem ist das World Geodetic System 1984 (WGS 84) Die Dateien sind nach dem Schema ILATM1B\_YYYYMMDD\_HHMMSS.ATM4BT4.h5 benannt:

- Datensatz-ID\_
- Jahr, Monat, Tag der Messung\_
- Zeitpunkt am Beginn der Messung in Stunden, Minuten, Sekunden\_
- Instrumenten-ID (fünfstellig) und Messwinkel (T2 = 15 Grad, T3 = 23 Grad, T4 = 30 Grad).
- xxx Dateityp (.h5 = HDF5, h5.xml = zusätzliche Metadatendatei)

Studinger, M. 2013, updated 2020. IceBridge ATM L1B Elevation and Return Strength, Version 2. [Indicate subset used]. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center. <https://doi.org/10.5067/19SIM5TXKPGT>. [05.12.2024]

#### **56.1.1 PyTables**

[HDFStore-Objekt](#)

•

•

•

•

**i** Note ???: inkompatible HDF5-Datei

```
# Datei öffnen
dateipfad = "01-daten/ILATM1B_20191120_041200.ATM6AT6.h5"
hdf = pd.HDFStore(dateipfad, mode = 'r')

print(hdf.info(), "\n")

# parameter include: str, default 'pandas'. When 'pandas' return pandas objects. When 'nat
print(hdf.keys(include = 'native'), "\n")
print(hdf.keys(include = 'pandas'), "\n")

print(hdf.groups(), "\n")

# Test hdf.get()
try:
    elevation = hdf.get('elevation')
    print(type(elevation))
except TypeError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    elevation = hdf.get('elevation')
    print(type(elevation))

# Test pd.read_hdf()
try:
    elevation = pd.read_hdf(path_or_buf = hdf, key = 'elevation')
    print(type(elevation))
except TypeError as error:
    print("Die Eingabe führt zu der Fehlermeldung:\n", error)
else:
    elevation = pd.read_hdf(path_or_buf = hdf, key = 'elevation')
    print(type(elevation))

# HDFStore-Objekt schließen
hdf.close()

<class 'pandas.io.pytables.HDFStore'>
File path: 01-daten/ILATM1B_20191120_041200.ATM6AT6.h5
Empty

[]
```

[]

[]

Die Eingabe führt zu der Fehlermeldung:

cannot create a storer if the object is not existing nor a value are passed

Die Eingabe führt zu der Fehlermeldung:

cannot create a storer if the object is not existing nor a value are passed

```
# HDF5-Datei anlegen
df = pd.DataFrame({'Spalte1': [1, 2, 3], 'Spalte2': [4, 5, 6]})
hdf = pd.HDFStore("01-daten/hdf_file_demo.h5", mode = 'a')
hdf.put('daten', df, format = 'table')

hdf.close()

# HDF5-Datei einlesen
hdf2 = pd.HDFStore("01-daten/hdf_file_demo.h5", mode = 'r')
print(hdf2.info(), "\n")
print(hdf2.keys(), "\n")

df2 = hdf2.get(key = "/daten")
print(df2)

hdf2.close()
```

### 56.1.2 h5py

siehe h5py Dokumentation

Dokumentation h5py

```
import h5py
dateipfad = "01-daten/ILATM1B_20191120_041200.ATM6AT6.h5"
hdf = h5py.File(dateipfad, mode = 'r')

print(hdf, "\n")
print(list(hdf.keys()))
```

```
for key in list(hdf.keys()):  
  
    object = hdf[key]  
    print(f"Schlüssel:{key} ist ein {type(object)}")  
  
    if isinstance(hdf[key], h5py.Dataset):  
  
        print(f"\tDatentyp: {object.dtype}\n"  
              f"\tStruktur: {object.shape}\t Dimensionen: {object.ndim}\tAnzahl Elemente: {object.size}")  
  
    elif isinstance(hdf[key], h5py.Group):  
        print(f"\tDie Gruppe {key} enthält die Objekte:\n\t {hdf[key].keys()}\n")
```

```
try:  
    print(hdf['elevation'] - 1000)  
except TypeError as error:  
    print("Der Direktzugriff führt zu der Fehlermeldung:\n", error)  
  
# Mit einer Kopie kann gearbeitet werden  
print("\nMit einer Kopie geht es:")  
print(hdf['elevation'][:] - 1000)  
  
# Mit Indexbereichen ebenso  
print("\nEbenso mit ausgewählten Indexbereichen:")  
print(hdf['elevation'][0:10])  
  
hdf.close()
```

### 56.1.3 Übung Zugriff auf H5P-Datasets

💡 Tip ??: Musterlösung absolute Zeit berechnen

```
# HDF-Datei öffnen
dateipfad = "01-daten/ILATM1B_20191120_041200.ATM6AT6.h5"
hdf = h5py.File(dateipfad, mode = 'r')

zeitstempel = int(str(hdf)[29:35])
print(f"Beginn der Messungen: {zeitstempel}\n")

rel_time = hdf['instrument_parameters/rel_time'][:]
abs_time = rel_time + zeitstempel

print(f"rel_time[-5:]:\n{rel_time[-5:]}\n\nabs_time[-5:]:\n{abs_time[-5:]}")

# HDF-Datei schließen
hdf.close()

Beginn der Messungen: 41200

rel_time[-5:]:
[59.986 59.986 59.987 59.989 59.99]

abs_time[-5:]:
[41259.984 41259.984 41259.99 41259.99 41259.99]
```

### 56.1.3.1 HDF5 Dateien schreiben

```
hdf_neu = h5py.File('01-daten/hdf_neu.h5', mode = 'w')
hdf_neu['abs_time'] = abs_time
hdf_neu.close()

# Kontrolle
hdf_neu = h5py.File('01-daten/hdf_neu.h5', mode = 'r')
print(hdf_neu.keys())
hdf_neu.close()
```

[Dokumentation h5py](#)

## **56.2 netCDF4**

- 
- 
- 
- 
- 
- 
- 
- 
- 

## **56.3 Datensatz**

kostenlose Registrierung bei NASA Earth erforderlich

Datensatzdokumentation Poetzsch 2021

- 
-

```
import netCDF4 as nc

dateipfad = "01-daten/LISOTD_HRFC_V2.3.2015.nc"
cdf = nc.Dataset(filename = dateipfad, mode = 'r')
print(f"Dateiversion: {cdf.data_model}\n")

print(cdf.groups)
```

```
variables = cdf.variables
for key, value in variables.items():
    print(key)
    print(value, "\n")
    break # Abbruch nach erstem Schlüssel-Wert-Paar
```

**i** Note ???: vollständige Ausgabe cdf.variables

```
variables = cdf.variables
for key, value in variables.items():
    print(key)
    print(value, "\n")

DE_By_Threshold
<class 'netCDF4.Variable'>
float32 DE_By_Threshold(DE_By_Threshold)
    long_name: By Threshold
    comment: Threshold index for OTD detection efficiency
    units: 1
unlimited dimensions:
current shape = (3,)
filling on, default _FillValue of 9.969209968386869e+36 used
```

```

DE_Local_Hour
<class 'netCDF4.Variable'>
float32 DE_Local_Hour(DE_Local_Hour)
    long_name: Local Hour
    comment: Lightning detection efficiency is the function of local hour. A day is divided into 24 hours.
    units: 1
unlimited dimensions:
current shape = (24,)
filling on, default _FillValue of 9.969209968386869e+36 used

DE_LowRes_Latitude
<class 'netCDF4.Variable'>
float32 DE_LowRes_Latitude(DE_LowRes_Latitude)
    long_name: LowRes Latitude
    standard_name: latitude
    axis: Y
    units: degrees_north
unlimited dimensions:
current shape = (72,)
filling on, default _FillValue of 9.969209968386869e+36 used

DE_LowRes_Longitude
<class 'netCDF4.Variable'>
float32 DE_LowRes_Longitude(DE_LowRes_Longitude)
    long_name: LowRes Longitude
    standard_name: longitude
    axis: X
    units: degrees_east
unlimited dimensions:
current shape = (144,)
filling on, default _FillValue of 9.969209968386869e+36 used

HRFC_AREA
<class 'netCDF4.Variable'>
float32 HRFC_AREA(Latitude, Longitude)
    valid_range: [ 13.48325 3090.078 ]
    long_name: Grid Cell Area
    units: km^2
    _FillValue: 0.0
unlimited dimensions:
current shape = (360, 720)

```

```

filling on

HRFC_COM_FR
<class 'netCDF4.Variable'>
float32 HRFC_COM_FR(Latitude, Longitude)
    valid_range: [ 0.      201.7618]
    long_name: Combined Flash Rate Climatology
    _FillValue: 0.0
    units: count/km^2/year
unlimited dimensions:
current shape = (360, 720)
filling on

HRFC_LIS_DE
<class 'netCDF4.Variable'>
float32 HRFC_LIS_DE(DE_Local_Hour, DE_LowRes_Latitude, DE_LowRes_Longitude)
    valid_range: [ 0.      88.00034]
    long_name: Applied LIS Detection Efficiency
    units: Percent
    _FillValue: 0.0
unlimited dimensions:
current shape = (24, 72, 144)
filling on

HRFC_LIS_FR
<class 'netCDF4.Variable'>
float32 HRFC_LIS_FR(Latitude, Longitude)
    valid_range: [ 0.      5723.275]
    long_name: LIS Flash Rate Climatology
    _FillValue: 0.0
    units: count/km^2/year
unlimited dimensions:
current shape = (360, 720)
filling on

HRFC_LIS_RF
<class 'netCDF4.Variable'>
float32 HRFC_LIS_RF(Latitude, Longitude)
    valid_range: [ 0.  9770.]
    long_name: LIS Raw Flashes
    _FillValue: 0.0

```

```

    units: count
unlimited dimensions:
current shape = (360, 720)
filling on

HRFC_LIS_SF
<class 'netCDF4.Variable'>
float32 HRFC_LIS_SF(Latitude, Longitude)
    valid_range: [ 0. 12500.17]
    long_name: LIS Scaled Flashes
    _FillValue: 0.0
    units: count
unlimited dimensions:
current shape = (360, 720)
filling on

HRFC_LIS_VT
<class 'netCDF4.Variable'>
float32 HRFC_LIS_VT(Latitude, Longitude)
    valid_range: [0.00000e+00 4.462968e+09]
    long_name: LIS Viewtime
    units: km^2 sec
    _FillValue: 0.0
unlimited dimensions:
current shape = (360, 720)
filling on

HRFC_OTD_DE
<class 'netCDF4.Variable'>
float32 HRFC_OTD_DE(DE_By_Threshold, DE_Local_Hour, DE_LowRes_Latitude, DE_LowRes_Longitude)
    valid_range: [ 0. 53.6012]
    long_name: Applied OTD Base Detection Efficiency
    units: Percent
    _FillValue: 0.0
unlimited dimensions:
current shape = (3, 24, 72, 144)
filling on

HRFC_OTD_FR
<class 'netCDF4.Variable'>
float32 HRFC_OTD_FR(Latitude, Longitude)

```

```
valid_range: [ 0.      201.7618]
long_name: OTD Flash Rate Climatology
_FILLValue: 0.0
units: count/km^2/year
unlimited dimensions:
current shape = (360, 720)
filling on

HRFC_OTD_RF
<class 'netCDF4.Variable'>
float32 HRFC_OTD_RF(Latitude, Longitude)
    valid_range: [ 0. 1298.]
    long_name: OTD Raw Flashes
    _FillValue: 0.0
    units: count
unlimited dimensions:
current shape = (360, 720)
filling on

HRFC_OTD_SF
<class 'netCDF4.Variable'>
float32 HRFC_OTD_SF(Latitude, Longitude)
    valid_range: [ 0.      2645.824]
    long_name: OTD Scaled Flashes
    _FillValue: 0.0
    units: count
unlimited dimensions:
current shape = (360, 720)
filling on

HRFC_OTD_VT
<class 'netCDF4.Variable'>
float32 HRFC_OTD_VT(Latitude, Longitude)
    valid_range: [0.00000e+00 1.119883e+09]
    long_name: OTD Viewtime
    units: km^2 sec
    _FillValue: 0.0
unlimited dimensions:
current shape = (360, 720)
filling on
```

```

Latitude
<class 'netCDF4.Variable'>
float32 Latitude(Latitude)
    long_name: Latitude
    standard_name: latitude
    axis: Y
    units: degrees_north
unlimited dimensions:
current shape = (360,)
filling on, default _FillValue of 9.969209968386869e+36 used

Longitude
<class 'netCDF4.Variable'>
float32 Longitude(Longitude)
    long_name: Longitude
    standard_name: longitude
    axis: X
    units: degrees_east
unlimited dimensions:
current shape = (720,)
filling on, default _FillValue of 9.969209968386869e+36 used

lis_rf_data_centered
<class 'netCDF4.Variable'>
float32 lis_rf_data_centered(latitude, longitude)
    description: zentrierte Blitzanzahl
    units: absolute Abweichung vom Mittelwert
unlimited dimensions:
current shape = (360, 720)
filling on, default _FillValue of 9.969209968386869e+36 used

```

## 56.4 Dimensionen

```
dimensions = cdf.dimensions

for key, value in dimensions.items():
    print(key)
    print(value, "\n")
```

## 56.5 globale Attribute

```
print(cdf.ncattrs(), "\n")
global_attributes = cdf.__dict__

for key, value in global_attributes.items():
    print(key)
    print(value, "\n")
```

## 56.6 Attribute einer Variablen

```
variable = cdf['HRFC_OTD_FR']
# Alternativ:
# variable = cdf.variables['HRFC_OTD_FR']
```

```
print("\nAttribute einer Variablen")
print(f"Als Liste mit variable.ncattrs():\n{variable.ncattrs()}\n")

print(f"Als Dictionary mit variable.__dict__:\n{variable.__dict__}\n")
```

### 56.6.1 Daten in netCDF4-Dateien schreiben

- 
- 
-

```
# Neues Dataset im Schreibmodus öffnen
cdf_neu = nc.Dataset('01-daten/cdf_neu.nc', mode = 'w')

# Gruppe 'Daten' anlegen
cdf_neu.createGroup("Daten")

# Dimension anlegen
cdf_neu.createDimension('abs_time', abs_time.shape[0])

# Variable anlegen als 'f4' 32-bit float und einem Objekt zuweisen.
variablename = cdf_neu.createVariable('/Daten/abs_time', 'f4', ('abs_time',))

## Attribute anlegen
variablename.description = 'sehr wichtige Zeitinformationen'
variablename.source = 'berechnet aus IceBridge ATM L1B Elevation and Return Strength, Version'
variablename.units = 'Millisekunden'
```

```

# Dataset schließen
cdf_neu.close()

# Kontrolle
cdf_neu = nc.Dataset('01-daten/cdf_neu.nc', mode = 'r')
print(cdf_neu.groups, "\n")

variables = cdf_neu['/Daten'].variables
for key, value in variables.items():
    print(key)
    print(value, "\n")

cdf_neu.close()

```

### 56.6.1.1 Operationen mit Variablen

```

try:
    print(cdf['HRFC_OTD_FR'] - 1)
except TypeError as error:
    print("Der Direktzugriff führt zu der Fehlermeldung:\n", error)

# Mit einer Kopie kann gearbeitet werden

```

```
print("\nMit einer Kopie geht es:")
print(cdf['HRFC_OTD_FR'][ :, :])

# Mit Indexbereichen ebenso
print("\nEbenso mit ausgewählten Indexbereichen:")
print(cdf['HRFC_OTD_FR'][30:40, 30:40])
```

```
cdf.close()
```

### 56.6.2 Übung Zugriff auf netCDF-Datasets

-

- 
- 

 Tip ??

Da die Ausführung des Codes die Datei entsprechend der Aufgabenstellung verändert, ist der Code nicht als Python-Code eingebunden.

```

# Datei im Modus append einlesen
dateipfad = "01-daten/LISOTD_HRFC_V2.3.2015.nc"
cdf = nc.Dataset(filename = dateipfad, mode = 'a')
print(cdf.data_model, "\n")

# Variable einlesen
lis_rf = cdf['HRFC_LIS_RF']

# Attribute der Variablen ausgeben
print("Attribute der Variablen 'HRFC_LIS_RF':", lis_rf.ncattrs(), "\n")

# Daten aus der Variablen auslesen
lis_rf_data = cdf['HRFC_LIS_RF'][:, :]
print(f"min: {lis_rf_data.min()}\tmax: {lis_rf_data.max()}\tmean: {lis_rf_data.mean()}\n")

# Daten zentrieren
# runden wegen interner Darstellung von Gleitkommazahlen - optional
lis_rf_data_centered = lis_rf_data - lis_rf_data.mean()
print(f"min: {lis_rf_data_centered.min()}\tmax: {lis_rf_data_centered.max()}\tmean: {lis_rf_data_centered.mean()}\n")

# Daten an die netCDF4-Datei anhängen

## Dimension anlegen
## latitude und longitude sind die Dimensionen des 2-dimensionalen Arrays
## Code kann nur einmal ausgeführt werden, prüfen mit try: legt die Dimension bereits an
## deshalb Konstruktion mit not in, um zu prüfen, ob Dimension bereits existiert

dimensions = cdf.dimensions

### latitude
DIMENSION = 'latitude'
if DIMENSION not in dimensions:
    print(f"Die Dimension {DIMENSION} existiert nicht und wird angelegt.")
    cdf.createDimension(DIMENSION, lis_rf_data_centered.shape[0])
else:
    print(f"Die Dimension {DIMENSION} existiert bereits.")

### longitude
DIMENSION = 'longitude'
if DIMENSION not in dimensions:
    print(f"Die Dimension {DIMENSION} existiert nicht und wird angelegt.")
    cdf.createDimension(DIMENSION, lis_rf_data_centered.shape[1])
else:
    print(f"Die Dimension {DIMENSION} existiert bereits.")

## Variable 'lis_rf_data_centered' anlegen als 'f4' 32-bit float und dem Objekt variablenan
## latitude und longitude sind die Dimensionen des 2-dimensionalen Arrays
## Code kann nur einmal ausgeführt werden, prüfen mit try: legt die Variable bereits an, da
variables = cdf.variables

VARIABLE = 'lis_rf_data_centered'

if VARIABLE not in variables:

```

## **57 Das Wichtigste**

# 58 Lernzielkontrolle

## 58.1 Kompetenzquiz

1.

```
import pandas as pd

dateipfad = '01-daten/quiz-aufgabe1.csv'
aufgabe1 = pd.read_csv(filepath_or_buffer = dateipfad)
print(aufgabe1.info())
```

2.

3.

4.

5.

6.

```
print(m_daten, "\n")  
  
m_daten.mask = ma.nomask  
print(m_daten, "\n")
```

7.

A)

B)

C)

D)

### 💡 Lösungen

Aufgabe 1: pd.read\_csv()

```

dateipfad = '01-daten/quiz-aufgabe1.csv'
aufgabe1 = pd.read_csv(filepath_or_buffer = dateipfad, sep = ';', parse_dates = ['Geburtsdatum'])
print(aufgabe1.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   ID          4 non-null      int64  
 1   Name         4 non-null      string  
 2   Geburtsdatum 4 non-null      datetime64[ns]
 3   Gehalt       4 non-null      int64  
dtypes: datetime64[ns](1), int64(2), string(1)
memory usage: 260.0 bytes
None

```

Aufgabe 2: Beobachtungen als ungültig markieren

```

bedingung = aufgabe1['Geburtsdatum'].dt.year < 1980
print(bedingung, "\n")

# pd.NA für Zeichenketten und Ganzzahlen
## NumPy-Datentyp int64 führt zur Umwandlung des Datentyps der Spalte Gehalt
aufgabe2 = aufgabe1.copy()
aufgabe2.loc[bedingung, :] = pd.NA
print("NumPy-Datentyp int64 führt zur Umwandlung des Datentyps der Spalte Gehalt")
print(aufgabe2, "\n")
# 3  NaN    <NA>        NaT        NaN

## Pandas-Datentyp Int64 unterstützt fehlende Werte für Ganzzahlen
aufgabe2 = aufgabe1.copy()
aufgabe2['Gehalt'] = aufgabe2['Gehalt'].astype('Int64')
aufgabe2['ID'] = aufgabe2['ID'].astype('Int64')
aufgabe2.loc[bedingung, :] = pd.NA
print("Pandas-Datentyp Int64 unterstützt fehlende Werte für Ganzzahlen")
print(aufgabe2)
# 3  NaN    <NA>        NaT        <NA>

0   False
1   False
2   False

```

```
3      True
Name: Geburtsdatum, dtype: bool
```

NumPy-Datentyp int64 führt zur Umwandlung des Datentyps der Spalte Gehalt

	ID	Name	Geburtsdatum	Gehalt
0	1.0	Anna	1986-04-12	55000.0
1	2.0	Bernd	1990-05-23	62000.0
2	3.0	Carla	1982-11-30	71000.0
3	NaN	<NA>	NaT	NaN

Pandas-Datentyp Int64 unterstützt fehlende Werte für Ganzzahlen

	ID	Name	Geburtsdatum	Gehalt
0	1	Anna	1986-04-12	55000
1	2	Bernd	1990-05-23	62000
2	3	Carla	1982-11-30	71000
3	<NA>	<NA>	NaT	<NA>

Aufgabe 3: datetime aufsteigend sortieren

```
aufgabe3 = aufgabe2.copy()
print(aufgabe3['Geburtsdatum'].sort_values(ascending = False), "\n")

# Sortieren des DataFrames
print(aufgabe3.sort_values(by = 'Geburtsdatum', ascending = False, inplace = True), "\n")
```

```
1    1990-05-23
0    1986-04-12
2    1982-11-30
3        NaT
Name: Geburtsdatum, dtype: datetime64[ns]
```

None

Aufgabe 4: Welches Datum ist größer und warum?

Python zählt die Zeit ausgehend von der sogenannten Epoche pd.to\_datetime(0). Jüngere Menschen wurden in größerem Abstand zur Epoche geboren.

```
aufgabe4 = pd.DataFrame({'Geburtsdatum': aufgabe3['Geburtsdatum'].sort_values(ascending = False),
aufgabe4['timedelta zur Epoche'] = aufgabe4['Geburtsdatum'] - pd.to_datetime(0)

print(aufgabe4)
```

```
Geburtsdatum timedelta zur Epoche
1 1990-05-23           7447 days
0 1986-04-12           5945 days
2 1982-11-30           4716 days
3             NaT          NaT
```

Aufgabe 5: glob

```
glob.glob(pathname = 'ordnerpfad' + '?????[1-2].*') oder
glob.glob(pathname = 'ordnerpfad' + 'Datei[1-2].*') oder
glob.glob(pathname = 'ordnerpfad' + '*[1-2].*')
```

Aufgabe 6: masked Array

Das maskierte Array hat eine hard mask.

```
m_daten.soften_mask()
m_daten.mask = ma.nomask
print(m_daten, "\n")
```

```
[1 2 3 4]
```

Aufgabe 7: HDF5 und netCDF4

Richtige Antworten: A) und D)

## 58.2 Übungsaufgaben

### 58.2.0.1 Ein sehr unordentlicher Datensatz

[bei Eurostatt verfügbaren](#)

[Direktlink auf XLSX-Datei](#)

- 
- 
-

### **58.2.0.2 Ein schwieriges Format**

<https://www-genesis.destatis.de/datenbank/online/table/31111-0006/sequenz=tabelleErgebnis&selectionname=31111-0006&zeitscheiben=1>

- 
-

## **Part IX**

# **m-datenfitting-und-optimierung**

# Methodenbaustein Datenfitting und Datenoptimierung



Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann, Maik Poetzsch und Sebastian Seipel. Methodenbaustein Datenfitting und Datenoptimierung von Marc Fehr und Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar auf [GitHub](#). Ausgenommen von der Lizenz sind alle Logos Dritter und anders gekennzeichneten Inhalte. 2025

[https://github.com/bausteine-  
der-datenanalyse/m-datenfitting-und-optimierung](https://github.com/bausteine-der-datenanalyse/m-datenfitting-und-optimierung)

# **Voraussetzungen**

- 
- 
- 
- 
-

## **Lernziele**

- 
- 
- 
-

## 59 Einleitung

```
import numpy as np
import numpy.polynomial.polynomial as poly
import matplotlib.pyplot as plt
```

1.

2.

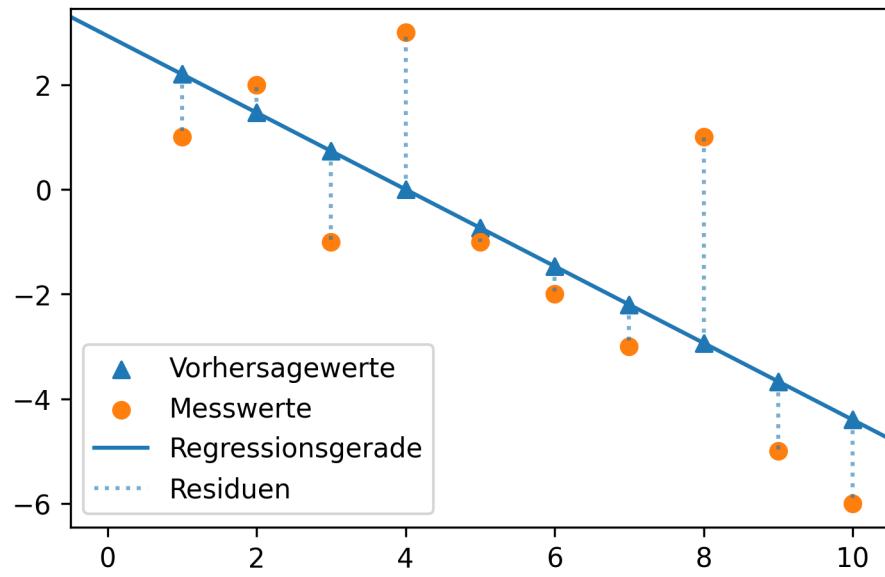
**i** Note ??: Methode der kleinsten Quadrate

Mit der Methode der kleinsten Quadrate soll diejenige Gerade  $\hat{y} = \beta_0 + \beta_1 \cdot x$  gefunden werden, die die quadrierten Abstände der Vorhersagewerte  $\hat{y}$  von den tatsächlich gemessenen Werten  $y$  minimiert. Die Werte  $y_i - \hat{y}_i$  sind die Residuen  $e_i$ . Es gilt also:

$$\sum_{i=1}^N (y_i - \hat{y}_i)^2 = \sum_{i=1}^N e_i = \min$$

Grafisch kann man sich die Minimierung der quadrierten Abstände so vorstellen.

## 59.1 Grafik



Regressionskoeffizienten: [ 2.93333333 -0.73333333]

## 59.2 Code

```

x = np.arange(1, 11)
y = - x.copy() + 4
y[0] -= 2
y[2] -= 2
y[3] += 3
y[-3] += 5

lm = poly.polyfit(x, y, 1)
vorhersagewerte = poly.polyval(x, lm)

plt.scatter(x, vorhersagewerte, label = 'Vorhersagewerte', marker = "^", color = "tab:blue")
plt.scatter(x, y, label = 'Messwerte', marker = 'o', color = "tab:orange")
plt.axline(xy1 = (0, lm[0]), slope = lm[1], label = "Regressionsgerade", color = "tab:blue")
dotted = plt.vlines(x, ymin = vorhersagewerte, ymax = y, alpha = 0.6, ls = 'dotted', label = "Residuen")

plt.legend()
plt.show()

print("Regressionskoeffizienten:", lm)

```

Die eingezeichnete Gerade entspricht der linearen Funktion  $\hat{y} = \beta_0 + \beta_1 \cdot x + e_i$ . Die Dreiecksmarker sind die Vorhersagewerte  $\hat{y}_i$  des linearen Modells für die Werte  $x_i = np.arange(1, 11)$ . Die tatsächlichen Messwerte  $y$  sind mit Kreismarkern markiert. Die Länge der gestrichelten Linien entspricht der Größe der Abweichung zwischen den Mess- und Vorhersagewerten  $y_i - \hat{y}_i$ , also den Residuen  $e_i$ .

Gesucht wird diejenige Gerade, die die Summe der quadrierten Residuen minimiert. Die gesuchten Werte  $\beta_0$  und  $\beta_1$  sind die Kleinst-Quadrat-Schätzer.

$$\beta_0 = \bar{y} - \beta_1 \cdot \bar{x}$$

$$\beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Der Vollständigkeit halber leiten wir die Kleinst-Quadrat-Schätzer her. Gesucht werden Werte für  $\beta_0$  und  $\beta_1$ , damit die Summe der Residuenquadrate  $\sum_{i=1}^n e_i^2$  möglichst klein wird. Die Residuenquadratsumme ist die Summe der quadrierten Differenzen aus beobachteten Werten  $y_i$  und der durch die lineare Funktion vorhergesagten Werte.

$$\sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i))^2$$

Wir untersuchen also eine Funktion, die von zwei Variablen abhängig ist.

$$f(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i))^2$$

Das Summenzeichen ist die Kurzschreibweise für eine Summe.

$$f(\beta_0, \beta_1) = (y_1 - (\beta_0 + \beta_1 \cdot x_1))^2 + (y_2 - (\beta_0 + \beta_1 \cdot x_2))^2 + \dots (y_n - (\beta_0 + \beta_1 \cdot x_n))^2$$

Im Minimum der Funktion müssen die beiden partiellen Ableitungen gleich Null sein (Warum das so ist, wird [hier](#) leicht verständlich erklärt.)

### Note 59.1: Partielle Ableitung

Die partielle Ableitung ist die Ableitung einer Funktion mit mehreren Variablen nach einer Variablen, wobei die übrigen Variablen als Konstanten behandelt werden.

Für eine Funktion  $f(x, y) = 2x + y^2$  wird die partielle Ableitung nach x so ausgedrückt:

$$\frac{\partial f(x,y)}{\partial x}$$

- Das Symbol ist die kursive Darstellung des kyrillischen Kleinbuchstaben (д) und wird als “del” gelesen. Es zeigt an, dass eine partielle Ableitung durchgeführt wird.
- Im Zähler steht die Funktion, die abgeleitet werden soll. Im Nenner steht die Variable nach der abgeleitet wird. Der Term wird gelesen als “del f von x und y nach del x”.

Die partielle Ableitung  $\frac{\partial f(x,y)}{\partial x} = 2$ .  $y^2$  wird als Konstante behandelt (z. B.  $5^2$ ) und ist abgeleitet Null.

Die partielle Ableitung  $\frac{\partial f(x,y)}{\partial y} = 2y$ .  $2x$  wird als Konstante behandelt (z. B.  $2 \cdot 3$ ) und ist abgeleitet Null.

In beiden partiellen Ableitungen sind  $x_i$  und  $y_i$  konstant. In der partiellen Ableitung nach  $\beta_0$  ist außerdem  $\beta_1$  konstant, in der partiellen Ableitung nach  $\beta_1$  ist entsprechend  $\beta_0$  konstant.

## 59.3 partielle Ableitung nach dem y-Achsenmittelpunkt

Für die partielle Ableitung nach  $\beta_0$  gilt also nach der Kettenregel für die äußere Funktion (oben) und die innere Funktion (Mitte):

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_0} = 2 \cdot (y_1 - (\beta_0 + \beta_1 \cdot x_1)) + \dots (y_n - (\beta_0 + \beta_1 \cdot x_n)) = 2 \cdot \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i)) \cdot (0 - (1 + 0 \cdot 0))$$

Für die partielle Ableitung nach  $\beta_0$  gilt also:

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_0} = -2 \cdot \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i)) = 0$$

Diese kann vereinfacht werden, indem der Vorfaktor  $-2$  entfällt (weil  $-2 \cdot 0 = 0$  gelten muss) und die Vorzeichen aufgelöst werden. Sodass:

$$\sum_{i=1}^n (y_i - \beta_0 - \beta_1 \cdot x_i) = 0$$

Man kann auch schreiben:

$$\sum_{i=1}^n y_i - \sum_{i=1}^n \beta_0 - \sum_{i=1}^n \beta_1 \cdot x_i = 0$$

$\beta_0$  und  $\beta_1$  sind Konstanten, sodass gilt  $\sum_{i=1}^n \beta_0 = \beta_0 \cdot \sum_{i=1}^n 1 = \beta_0 \cdot n$  und  $\sum_{i=1}^n \beta_1 \cdot x_i = \beta_1 \cdot \sum_{i=1}^n 1 \cdot x_i$ . So gilt:

$$\sum_{i=1}^n y_i - n \cdot \beta_0 - \beta_1 \cdot \sum_{i=1}^n x_i = 0$$

Jetzt kann man durch  $n$  teilen. Dabei entspricht  $\frac{\sum_{i=1}^n y_i}{n}$  dem arithmetischen Mittelwert von  $y$  und  $\frac{\sum_{i=1}^n x_i}{n}$  dem arithmetischen Mittelwert von  $x$ . Somit steht:

$$\bar{y} - \beta_0 - \beta_1 \cdot \bar{x} = 0$$

Umgestellt:

$$\beta_0 = \bar{y} - \beta_1 \cdot \bar{x}$$

## 59.4 partielle Ableitung nach dem Anstieg

Für die partielle Ableitung nach  $\beta_1$  ist ebenfalls die Kettenregel anzuwenden, sodass die äußere Funktion (oben) identisch abgeleitet wird:

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_1} = 2 \cdot (y_1 - (\beta_0 + \beta_1 \cdot x_1)) + \dots + (y_n - (\beta_0 + \beta_1 \cdot x_n)) = 2 \cdot \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i)) \cdot (0 - (0 + 1 \cdot x_i))$$

Für die partielle Ableitung nach  $\beta_1$  gilt also:

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_1} = -2 \sum_{i=1}^n x_i \cdot (y_i - (\beta_0 + \beta_1 \cdot x_i)) = 0$$

Auch diese kann vereinfacht werden, indem der Vorfaktor  $-2$  entfällt (weil  $-2 \cdot 0 = 0$  gelten muss) und die Vorzeichen aufgelöst werden. Außerdem kann ausmultipliziert werden:

$$\sum_{i=1}^n x_i y_i - \sum_{i=1}^n \beta_0 \cdot x_i - \sum_{i=1}^n \beta_1 \cdot x_i x_i = 0$$

Wieder können die Konstanten herausgezogen werden:

$$\sum_{i=1}^n x_i y_i - \beta_0 \cdot \sum_{i=1}^n x_i - \beta_1 \cdot \sum_{i=1}^n x_i x_i = 0$$

Jetzt kann man  $\beta_0 = \bar{y} - \beta_1 \cdot \bar{x}$  und  $\sum_{i=1}^n x_i = n \cdot \bar{x}$  einsetzen:

$$\sum_{i=1}^n x_i y_i - (\bar{y} - \beta_1 \cdot \bar{x}) \cdot n \cdot \bar{x} - \beta_1 \cdot \sum_{i=1}^n x_i x_i = 0$$

Der mittlere Term wird ausmultipliziert und  $x_i x_i$  im letzten Term als  $x_i^2$  geschrieben:

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} - \beta_1 \cdot n \bar{x} \bar{x} - \beta_1 \cdot \sum_{i=1}^n x_i^2 = 0$$

Die letzten beiden Terme werden unter Anwendung des Distributivgesetzes  $a - b = -(b - a)$  zusammengefasst.

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} - \beta_1 \cdot \left( \sum_{i=1}^n x_i^2 - n \bar{x}^2 \right) = 0$$

Jetzt kann nach  $\beta_1$  umgestellt werden. Erst:

$$\beta_1 \cdot \left( \sum_{i=1}^n x_i^2 - n \bar{x}^2 \right) = \sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}$$

Dann:

$$\beta_1 = \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{\sum_{i=1}^n x_i^2 - n \bar{x}^2}$$

Nun kann zuerst mit  $\sum_{i=1}^n x_i^2 - n \bar{x}^2 = \sum_{i=1}^n (x_i - \bar{x})^2$  umgeformt werden.

$$\beta_1 = \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Dann - und das wird gleich gezeigt - mit  $\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} = \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$ . Sodass steht:

$$\beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Der letzte Schritt wird ausgehend vom Ergebnis gezeigt und beginnt mit dem Ausmultiplizieren:

$$\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^n (x_i y_i - x_i \bar{y} - \bar{x} y_i + \bar{x} \bar{y})$$

Man kann auch schreiben:

$$\sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \bar{y} - \sum_{i=1}^n \bar{x} y_i + \sum_{i=1}^n \bar{x} \bar{y}$$

$\bar{x}$  und  $\bar{y}$  sind Konstanten, sodass  $\bar{x} \cdot \sum_{i=1}^n y_i$  und  $\bar{y} \cdot \sum_{i=1}^n x_i$  geschrieben werden kann.  $\sum_{i=1}^n x_i$  ist gleich  $n \cdot \bar{x}$  (analog für  $y$ ). So ergibt sich:

$$\sum_{i=1}^n x_i y_i - \bar{y} \cdot n \cdot \bar{x} - \bar{x} \cdot n \cdot \bar{y} + \sum_{i=1}^n \bar{x} \bar{y}$$

Sortieren:

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} - n \bar{x} \bar{y} + \sum_{i=1}^n \bar{x} \bar{y}$$

Der letzte Term  $\sum_{i=1}^n \bar{x} \bar{y}$  kann auch  $n \cdot \bar{x} \bar{y}$  geschrieben werden, sodass sich ergibt:

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} - n \bar{x} \bar{y} + n \bar{x} \bar{y}$$

Die letzten beiden Terme entfallen somit und es bleibt:

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}$$

[@Baitsch-2019, S. 73-74]

- 1.
- 2.
- 3.

# 60 Interpolation – Lücken schließen

## i Theorie - Modellierung

Die Modellierung von Daten hat das Ziel, eine Menge von Daten durch einen funktionalen Zusammenhang abzubilden. Daten aus Experimenten oder Simulationen können stark verrauscht und so für eine Weiterverarbeitung nicht geeignet sein. Eine mittelnde Funktion kann den Datensatz stark vereinfachen. Oder es existieren nur wenige Datenpunkte und die Zwischenstellen müssen durch eine Funktion bestimmt werden.

Generell kann die Modellierung von Daten auf folgendes Problem verallgemeinert werden:

1. Gegeben sind  $n$  Messpunktspaare  $(x_i, y_i)$  mit  $x_i, y_i \in \mathbb{R}$
2. Gesucht ist eine Modellfunktion  $y(x)$ , welche die Messpunktspaare approximiert

Ein möglicher Ansatz ist die Darstellung der Modellfunktion als Summe von  $m$  Basisfunktionen  $\phi_i(x)$  mit den Koeffizienten  $\beta_i$ .

$$y(x) = \sum_{i=1}^m \beta_i \cdot \phi_i(x) = \beta_1 \cdot \phi_1(x) + \cdots + \beta_m \cdot \phi_m(x)$$

Die Koeffizienten  $\beta_i$  müssen dabei so bestimmt werden, dass  $y(x)$  so gut wie möglich – oder gar exakt – die Messpunkte approximieren.

Als Abstandmaß zwischen einer Modellfunktion und den Messpunkten kann die [L2-Norm](#) verwendet werden. Diese ist definiert als

$$\|y(x) - (x_i, y_i)\|_2 = \sqrt{\sum_{i=1}^n (y(x_i) - y_i)^2} .$$

Eine solche Norm gibt ein Maß für die Qualität einer Approximation: je kleiner der Abstand, desto besser die Qualität. Dies ermöglicht das Finden optimaler Koeffizienten

und wird beispielsweise in der Methode der kleinsten Quadrate genutzt, in der ein Satz an Koeffizienten gesucht wird, der die L2-Norm minimiert.

## 60.1 Übersicht

- 
- 
- Taylor-  
Entwicklung
- 

## 60.2 Polynome

- 
- komplexe

```
[a0, a1, a2, ..., an]
```

```
P = np.array([3, -2, 5, 1])
print(P)
```

```
import numpy.polynomial.polynomial as poly
```

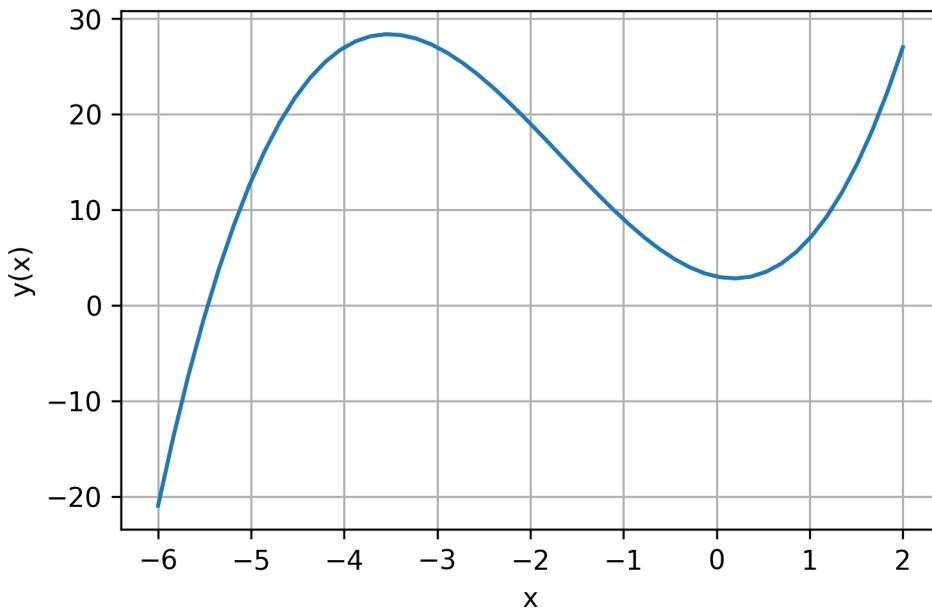
```
x = 1
y = poly.polyval(x, P)
print(f"P(x={x}) = {y}")
```

```
x = np.array([-1, 0, 1])
y = poly.polyval(x, P)
print(f"P(x={x}) = {y}")
```

```
x = np.linspace(-6, 2, 50)
y = poly.polyval(x, P)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y(x)')
plt.grid()

plt.show()
```



```
nullstellen = poly.polyroots(P)

# direkte Ausgabe des Arrays
print("Nullstellen: ")
print(nullstellen)
```

```
print("Nullstellen: ")
# schönere Ausgabe des Arrays
for i, z in enumerate(nullstellen):
    if z.imag == 0:
        print(f"  x_{i+1} = {z.real:.2}")
    else:
        print(f"  x_{i+1} = {z.real:.2} {z.imag:+.2}i")
```

```
nullstellen = [1, 2]
koeffizienten = poly.polyfromroots(nullstellen)

print("Nullstellen:", nullstellen)
print("Koeffizienten:", koeffizienten)
```

numpy-

Dokumentation

## 60.3 Polynominterpolation

```
# Beispieldaten aus y(x) = 2 - x

N = 50
dx = 0.25

def fnk(x):
    return 2 - x

x = np.array([1, 2])
y = fnk(x)

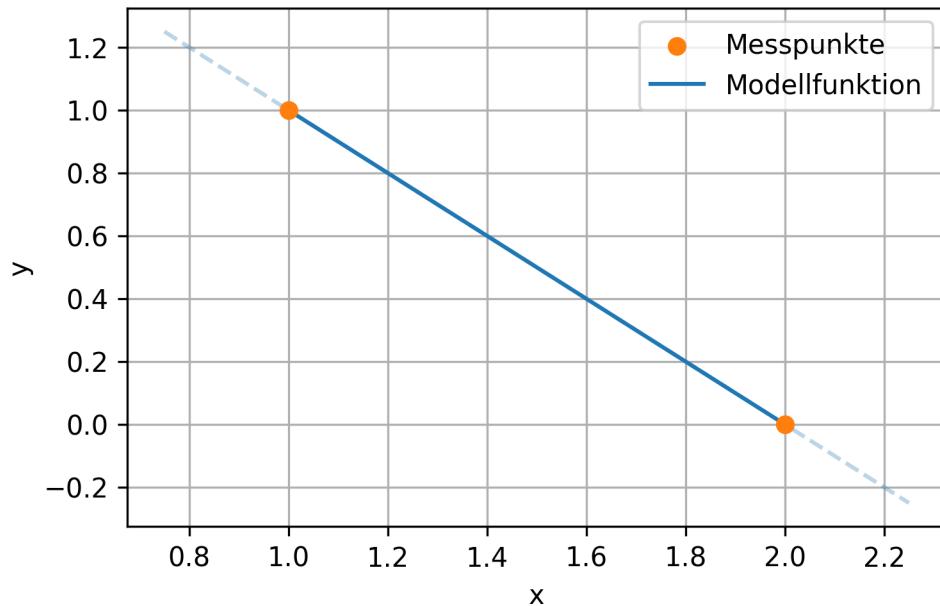
plt.scatter(x, y, color='C1', label="Messpunkte", zorder=3)

x_modell = np.linspace(np.min(x), np.max(x), N)
plt.plot(x_modell, fnk(x_modell), color='C0', label="Modellfunktion")

x_linie = np.linspace(np.min(x)-dx, np.max(x)+dx, N)
plt.plot(x_linie, fnk(x_linie), '--', alpha=0.3, color='C0')

plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid()

plt.show()
```



```
# Beispieldaten aus  $y(x) = -1 - 4x + 3x^2$ 

N = 50
dx = 0.25

def fnk(x):
    return -1 - 4*x + 3*x**2

x = np.array([-1, 2, 3])
y = fnk(x)

plt.scatter(x, y, color='C1', label="Messpunkte", zorder=3)

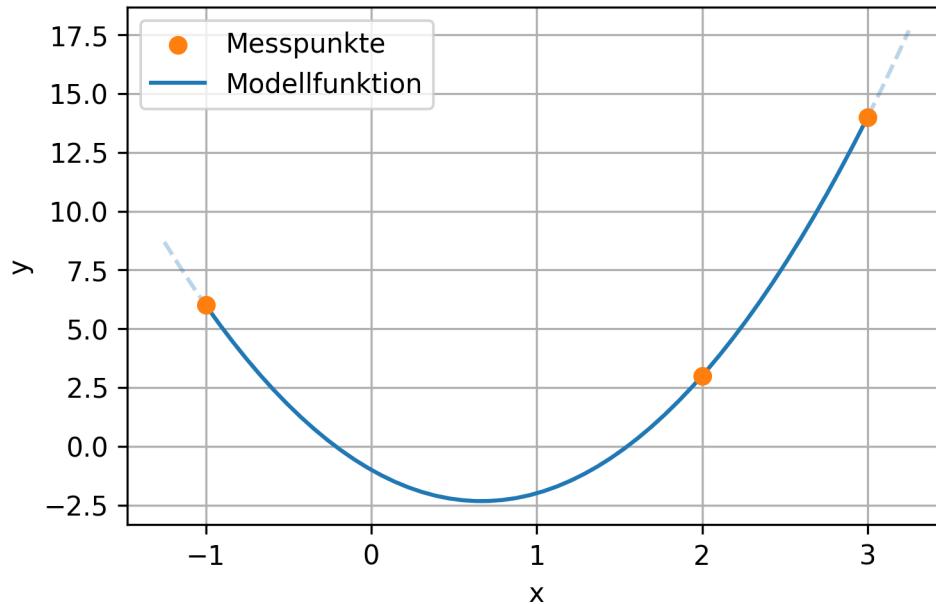
x_modell = np.linspace(np.min(x), np.max(x), N)
plt.plot(x_modell, fnk(x_modell), color='C0', label="Modellfunktion")

x_linie = np.linspace(np.min(x)-dx, np.max(x)+dx, N)
```

```
plt.plot(x_linie, fnk(x_linie), '--', alpha=0.3, color='CO')

plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid()

plt.show()
```



Vandermonde-Matrix

Funktion `numpy.polynomial.polynomial.polyfit()`

```
def fnk(x):
    return 0.5 + 1/(1+x**2)
```

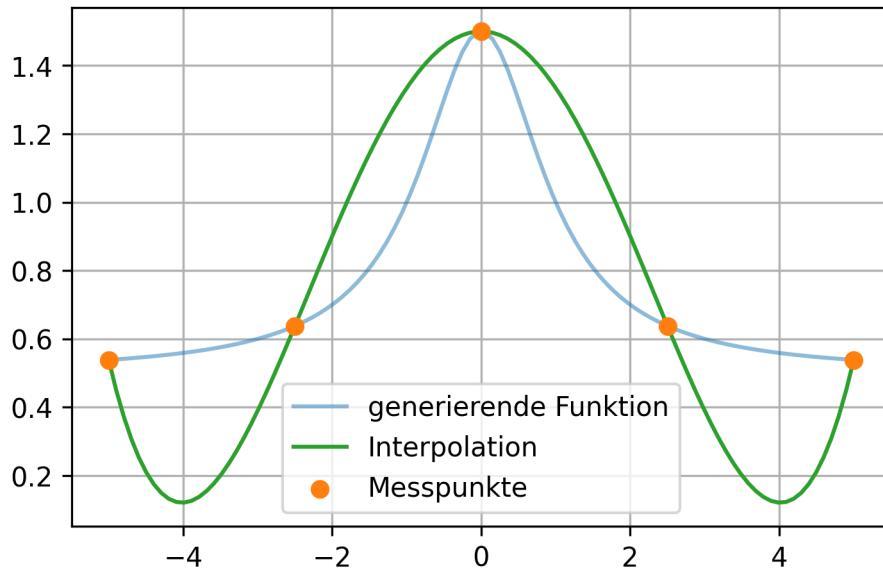
```
xmin = -5
xmax = 5
x = np.linspace(xmin, xmax, 100)
y = fnk(x)
```

```
n = 5
xi = np.linspace(xmin, xmax, n)
yi = fnk(xi)
```

```
P = poly.polyfit(xi, yi, n-1)
print("Interpolationskoeffizienten:")
print(P)
```

```
plt.plot(x, y, color='C0', alpha=0.5, label='generierende Funktion')
plt.plot(x, poly.polyval(x, P), color='C2', label='Interpolation')
plt.scatter(xi, yi, color='C1', label='Messpunkte', zorder=3)
plt.legend()
plt.grid()

plt.show()
```



```

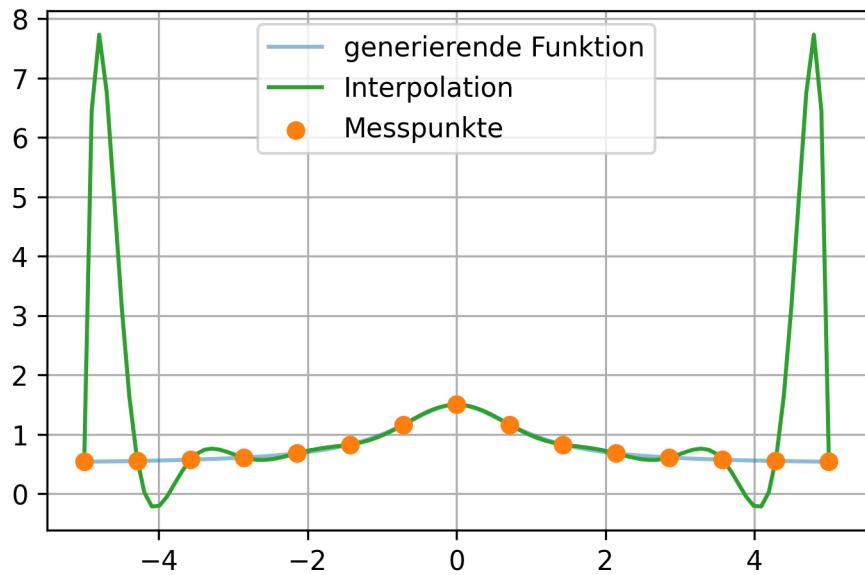
n = 15
xi = np.linspace(xmin, xmax, n)
yi = fnk(xi)

P = poly.polyfit(xi, yi, n-1)

plt.plot(x, y, color='C0', alpha=0.5, label='generierende Funktion')
plt.plot(x, poly.polyval(x, P), color='C2', label='Interpolation')
plt.scatter(xi, yi, color='C1', label='Messpunkte', zorder=3)
plt.legend()
plt.grid()

plt.show()

```



# 61 Fitting

```
xmin = 0
xmax = 5
x = np.linspace(xmin, xmax, 100)

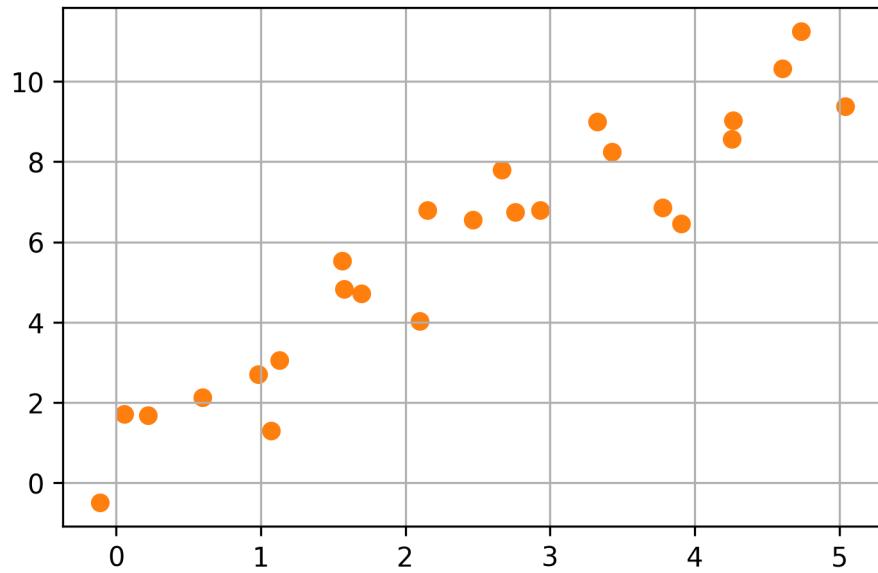
ni = 25

# x-Werte mit leichtem Rauschen
xi = np.linspace(xmin, xmax, ni) + 0.2*(2 * np.random.random(ni) -1)

# y(x) = 2x+0.5 mit leichtem Rauschen
yi = 2*xi + 0.5 + 2*(2 * np.random.random(ni) -1)

plt.scatter(xi, yi, color='C1')
plt.grid()

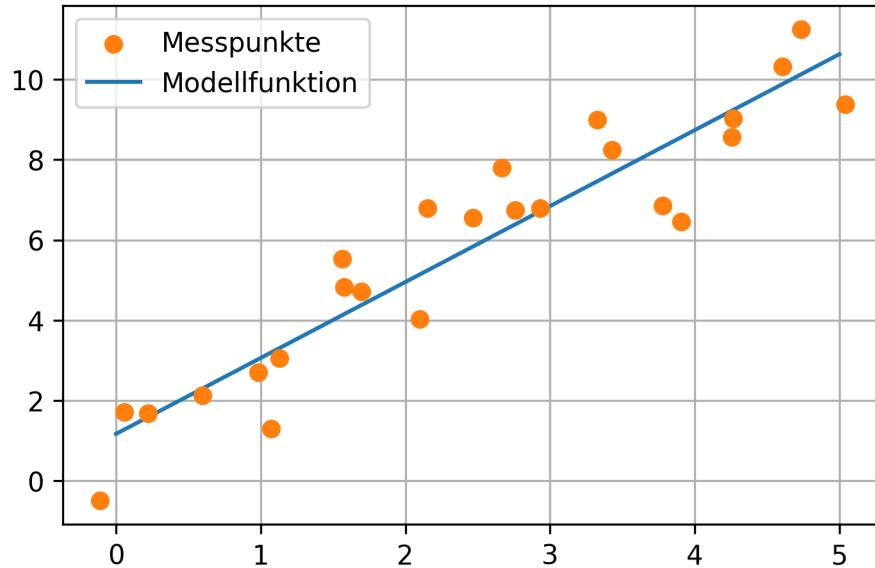
plt.show()
```



```
P1 = poly.polyfit(xi, yi, 1)

plt.scatter(xi, yi, color='C1', zorder=3, label='Messpunkte')
plt.plot(x, poly.polyval(x, P1), color='C0', label="Modellfunktion")
plt.grid()
plt.legend()

plt.show()
```



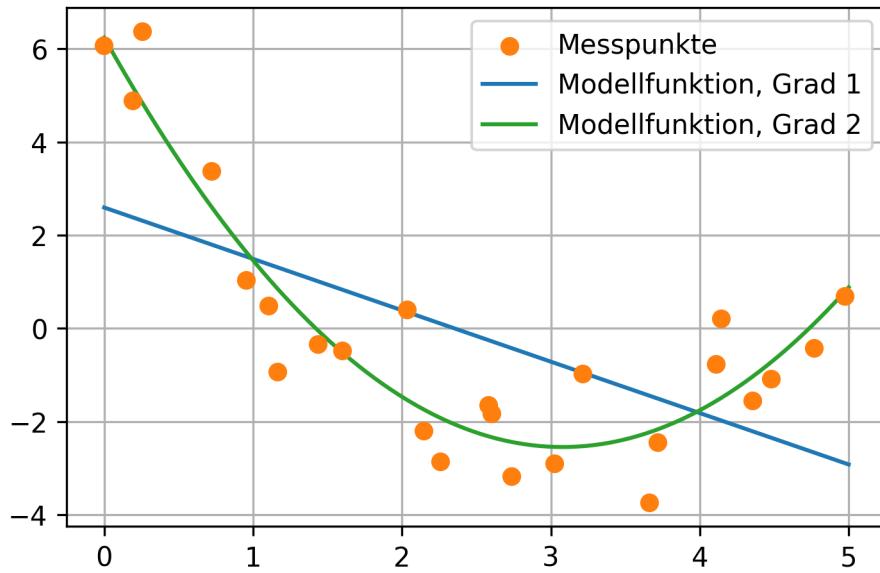
```
# x-Werte mit leichtem Rauschen
xi = np.linspace(xmin, xmax, ni) + 0.2*(2 * np.random.random(ni) - 1)
```

```
# y(x) = 2x+0.5 mit leichtem Rauschen
yi = (xi - 2)**2 - 2*xi + 2.5 + 2*(2 * np.random.random(ni) - 1)
```

```
P1 = poly.polyfit(xi, yi, 1)
P2 = poly.polyfit(xi, yi, 2)

plt.scatter(xi, yi, color='C1', zorder=3, label='Messpunkte')
plt.plot(x, poly.polyval(x, P1), color='C0', label="Modellfunktion, Grad 1")
plt.plot(x, poly.polyval(x, P2), color='C2', label="Modellfunktion, Grad 2")
plt.grid()
plt.legend()

plt.show()
```



# **62 Splines**

## **62.1 Definition**

•  
•  
•

•  
•

## **62.2 Kubische Splines**

•  
•  
•

- 
- 
- 
- 
- 
- 
- 

## 62.3 Anwendung

```
# Erzeugung von Messpunkten
n = 7
xi = np.linspace(0, np.pi, n)
yi = np.sin(xi)
```

```

# Wertebereich für die Visualisierung der Interpolation
x = np.linspace(0, np.pi, n*6)
y = np.sin(x)

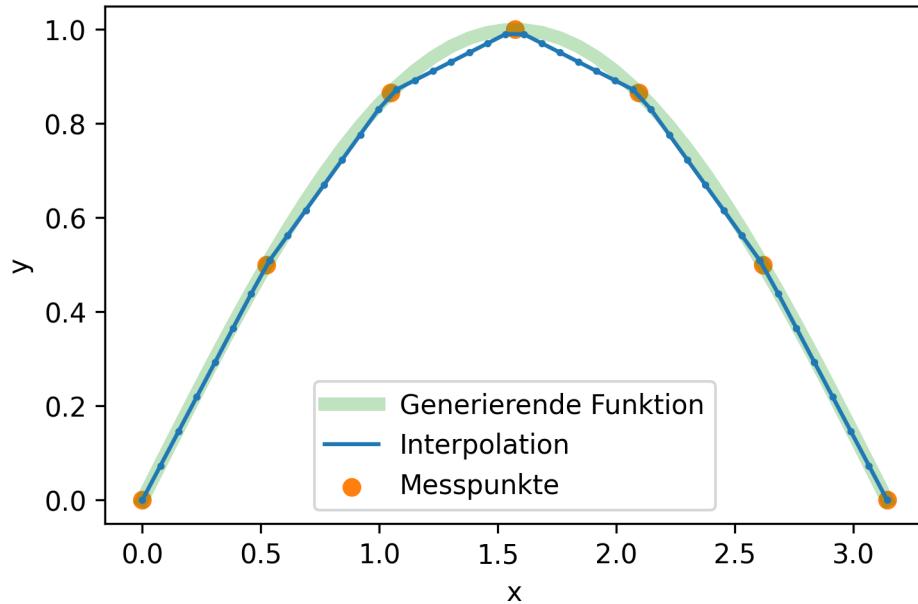
# Interpolation
y_s1 = np.interp(x, xi, yi)

plt.plot(x,y, alpha=0.3, color='C2', lw=5,
          label='Generierende Funktion')
plt.plot(x, y_s1, color='C0', label='Interpolation')
plt.scatter(x, y_s1, s=3, zorder=3, color='C0')
plt.scatter(xi, yi, color='C1', label='Messpunkte')

plt.xlabel('x')
plt.ylabel('y')
plt.legend();

plt.show()

```



```

import scipy.interpolate as si

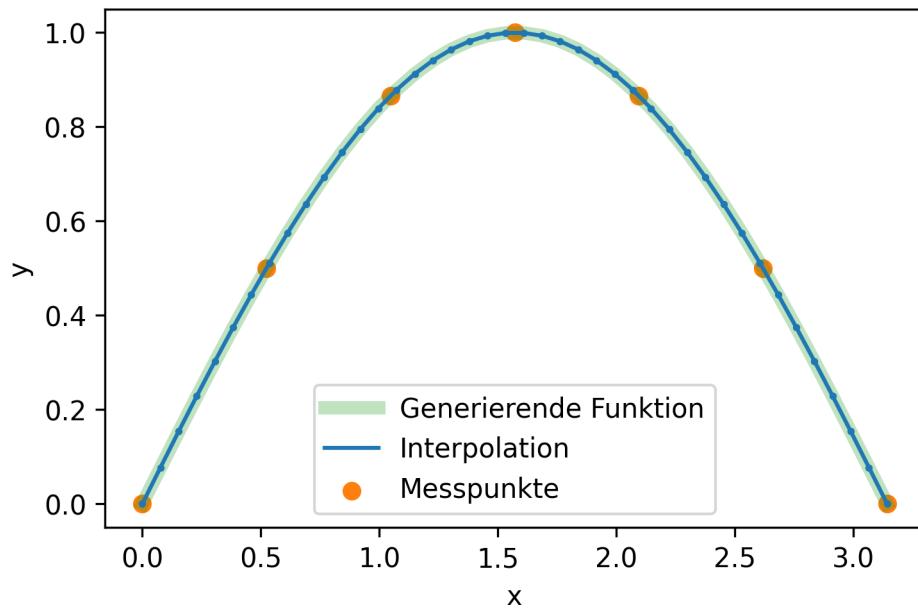
s3 = si.splrep(xi, yi)
y_s3 = si.splev(x, s3)

plt.plot(x,y, alpha=0.3, color='C2', lw=5,
          label='Generierende Funktion')
plt.plot(x, y_s3, color='C0', label='Interpolation')
plt.scatter(x, y_s3, s=3, zorder=3, color='C0')
plt.scatter(xi, yi, color='C1', label='Messpunkte')

plt.xlabel('x')
plt.ylabel('y')
plt.legend();

plt.show()

```



## 63 Trendglättung – Rauschen reduzieren

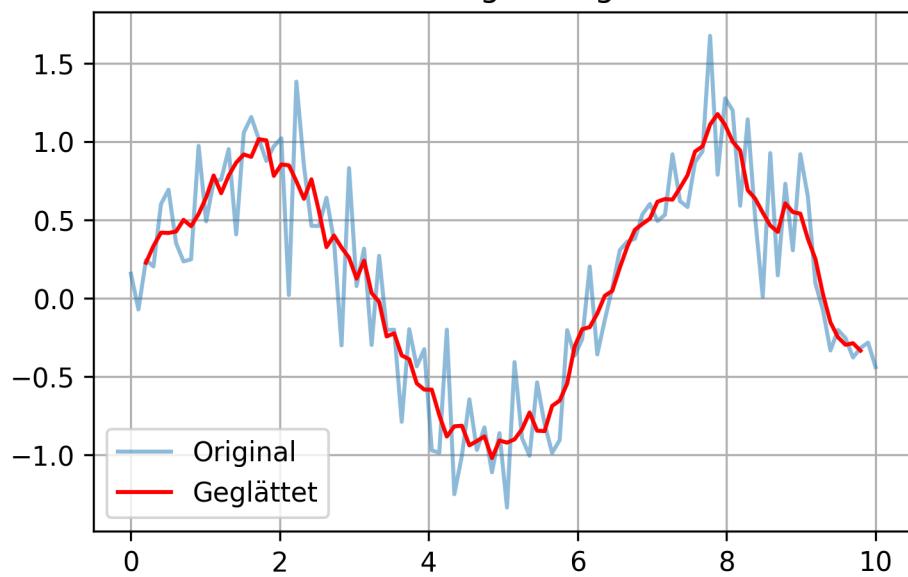
```
data = np.genfromtxt("01-daten/trenddaten-mit-rauschen.csv", delimiter=",", skip_header=1)
x = data[:, 0]
y = data[:, 1]

window = 5
weights = np.ones(window) / window
y_smooth = np.convolve(y, weights, mode='valid')

plt.plot(x, y, label="Original", alpha=0.5)
plt.plot(x[(window-1)//2:-(window//2)], y_smooth, label="Geglättet", color='red')
plt.legend()
plt.grid(True)
plt.title("Trendglättung")

plt.show()
```

Trendglättung

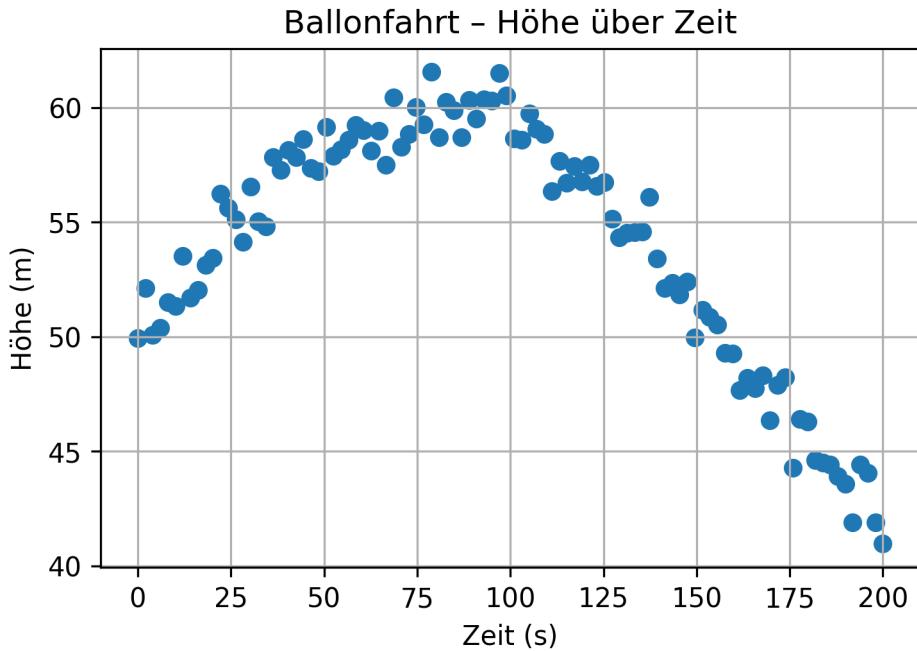


# 64 Übungen

## 64.1 Übung: Ballonfahrt-Daten analysieren

```
# Dateikopf als String einlesen
ballon = np.genfromtxt("01-daten/messdaten-ballonfahrt.txt", delimiter=",", max_rows= 4, dtype=None)

print("Der Kopf der Datei:")
print(ballon)
```



- 
- 

💡 Tip ??: Musterlösung Ballonfahrt

Zuerst werden die Daten ausgelesen.

```
# Daten auslesen
ballon = np.genfromtxt("01-daten/messdaten-ballonfahrt.txt", delimiter=",", skip_header=1)
zeit = ballon[:, 0]
hoehe = ballon[:, 1]
```

Anschließend berechnen wir die Steiggeschwindigkeit  $\frac{\Delta H_{hoe}}{\Delta Zeit}$ . Dazu verwenden wir die Funktion `np.diff()`. Diese berechnet die Differenz jedes Werts zu seinem Vorgänger `ergebnis[i] = wert[i+1] - wert[i]`. Für den i-ten Wert wird also keine Differenz

berechnet und das Ergebnis ist um ein Element kürzer. Die Steiggeschwindigkeit in  $\frac{m}{s}$  ergibt sich aus dem Quotienten beider Reihen.

```
# Differenzen berechnen
delta_hoehe = np.diff(hoehe)
print("Veränderung der Höhe: ", delta_hoehe[0:4])

delta_zeit = np.diff(zeit)
print("Veränderung der Zeit: ", delta_zeit[0:4])

# Steiggeschwindigkeit berechnen
steiggeschwindigkeit = delta_hoehe / delta_zeit
print("Steiggeschwindigkeit: ", steiggeschwindigkeit[0:4])
```

```
Veränderung der Höhe: [ 2.18551481 -2.05482638  0.32026988  1.12796595]
Veränderung der Zeit: [2.02020202 2.02020202 2.02020202 2.02020202]
Steiggeschwindigkeit: [ 1.08182983 -1.01713906  0.15853359  0.55834314]
```

Mit `numpy.polynomial.polynomial.polyfit()` berechnen wir erst ein Polynom dritten Grades und dann mit `numpy.polynomial.polynomial.polyval()` die gefitteten Daten.

```
polynom3_steiggeschwindigkeit = poly.polyfit(zeit[:-1], steiggeschwindigkeit, deg = 3)

fit_steiggeschwindigkeit = poly.polyval(x = zeit[:-1], c = polynom3_steiggeschwindigkeit)
```

```

# plotten
plt.suptitle("Ballonfahrt")

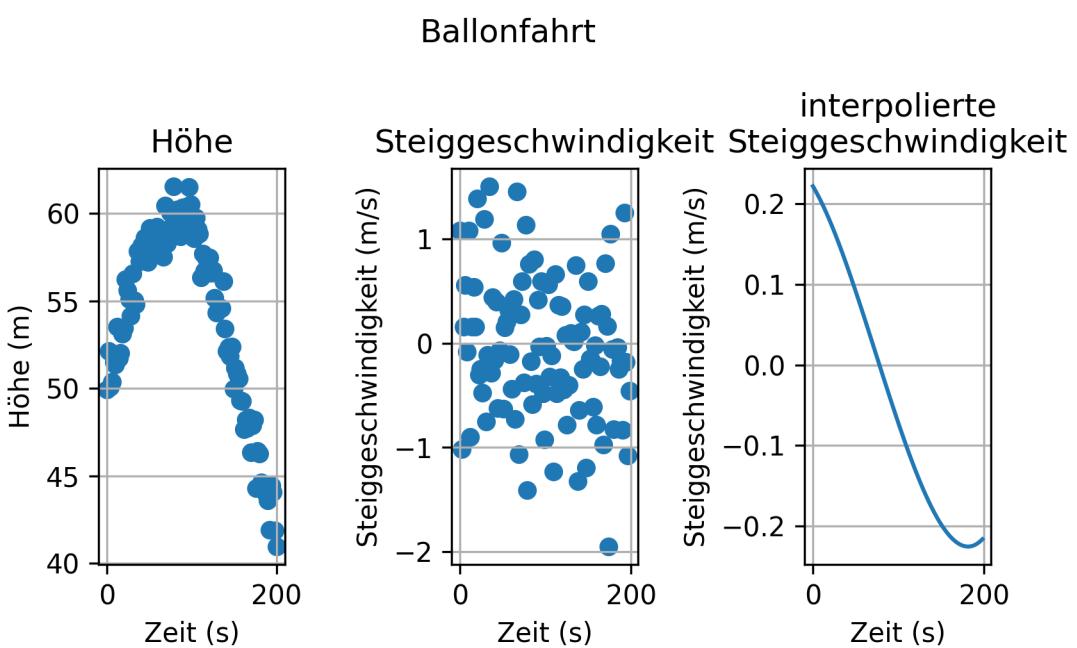
## subplot Höhe über Zeit
plt.subplot(1, 3, 1)
plt.scatter(zeit, hoehe)
plt.title("Höhe")
plt.xlabel("Zeit (s)")
plt.ylabel("Höhe (m)")
plt.grid(True)

## subplot Steiggeschwindigkeit über Zeit
## Länge des Arrays Steiggeschwindigkeit ist n-1
plt.subplot(1, 3, 2)
plt.scatter(zeit[:-1], steiggeschwindigkeit)
plt.title("Steiggeschwindigkeit")
plt.xlabel("Zeit (s)")
plt.ylabel("Steiggeschwindigkeit (m/s)")
plt.grid(True)

## subplot Polynom 3. Grades Steiggeschwindigkeit über Zeit
## Länge des Arrays Steiggeschwindigkeit ist n-1
plt.subplot(1, 3, 3)
plt.plot(zeit[:-1], fit_stieggeschwindigkeit)
plt.title("interpolierte\nSteiggeschwindigkeit")
plt.xlabel("Zeit (s)")
plt.ylabel("Steiggeschwindigkeit (m/s)")
plt.grid(True)

plt.tight_layout()
plt.show()

```



## 64.2 Übung: Balkenverformung im Bauingenieurwesen



Figure 64.1: Beispiel für einen Versuchsaufbau Balkenverformung

UniBw M

```
balken = np.genfromtxt("01-daten/balken-durchbiegung.csv", delimiter=",", max_rows= 4, dtype=
```

```
print("Der Kopf der Datei:")
print(balken)
```

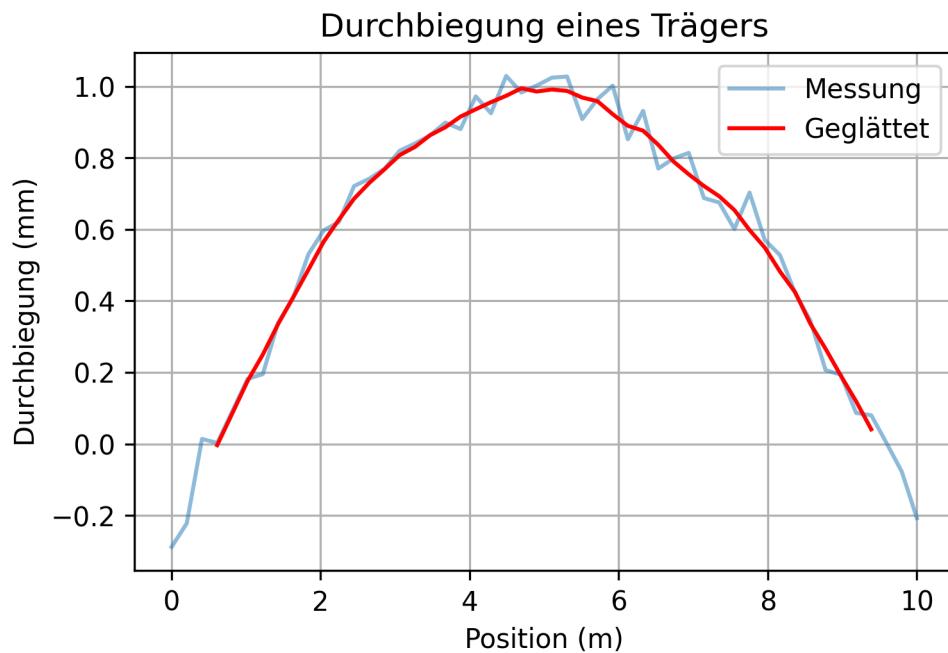
💡 Tip ??: Musterlösung Balken

```
balken = np.genfromtxt("01-daten/balken-durchbiegung.csv", delimiter=",", skip_header=1)
x = balken[:, 0]
y = balken[:, 1]

window = 7
weights = np.ones(window) / window
y_smooth = np.convolve(y, weights, mode='valid')

plt.plot(x, y, label="Messung", alpha=0.5)
plt.plot(x[(window-1)//2:-((window//2)]], y_smooth, label="Geglättet", color='red')
plt.title("Durchbiegung eines Trägers")
plt.xlabel("Position (m)")
plt.ylabel("Durchbiegung (mm)")
plt.legend()
plt.grid(True)

plt.show()
```



## 64.3 Übung: Neutronenstreuung

```
# Dateikopf als String einlesen
neutronen = np.genfromtxt("01-daten/neutronen.txt", delimiter="\t", max_rows= 4, dtype='str')
print(neutronen)
```

- 1.
- 2.
- 3.
- 4.
- 5.

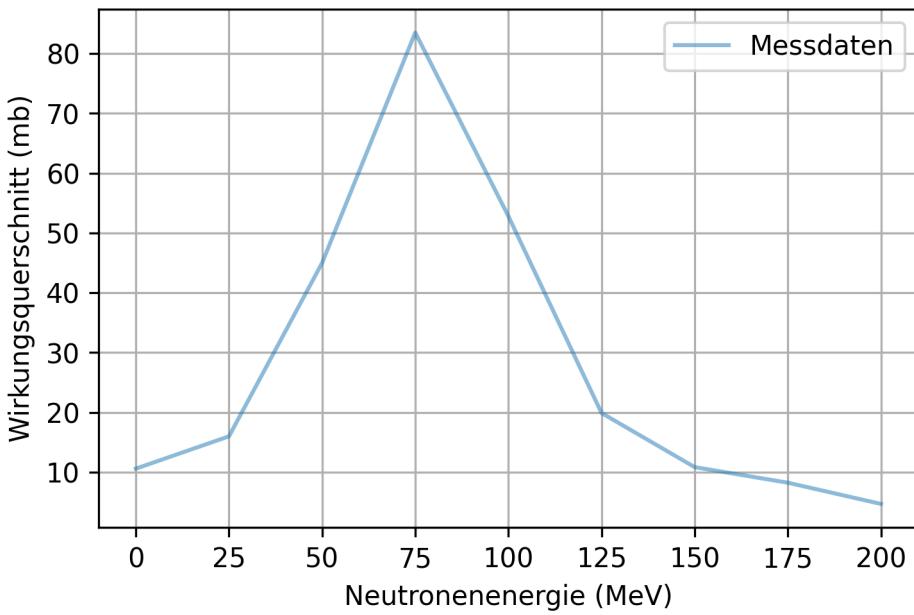
💡 Tip ???: Musterlösung Neutronenstreuung

1. Daten einlesen

```
neutronen = np.genfromtxt("01-daten/neutronen.txt", delimiter="\t", skip_header=1)
neutronenenergie = neutronen[:, 0]
wirkungsquerschnitt = neutronen[:, 1]

plt.plot(neutronenenergie, wirkungsquerschnitt, label="Messdaten", alpha=0.5)
plt.xlabel("Neutronenenergie (MeV)")
plt.ylabel("Wirkungsquerschnitt (mb)")
plt.legend()
plt.grid(True)

plt.show()
```



2. • 3. Polynom und Splines fitten:

```
# Polynom
polynom5 = poly.polyfit(neutronenenergie, wirkungsquerschnitt, deg = 5)
fit = poly.polyval(x = neutronenenergie, c = polynom5)

# Splines mit scipy
splines3 = si.splrep(neutronenenergie, wirkungsquerschnitt)
y_splines3 = si.splev(neutronenenergie, splines3)

# Splines mit NumPy
x_neu = np.linspace(neutronenenergie.min(), neutronenenergie.max(), num=50)
y_splines1 = np.interp(x_neu, neutronenenergie, wirkungsquerschnitt) # linear interpolieren
```

4. Grafische Darstellung:

```

plt.figure(figsize = [7, 5])

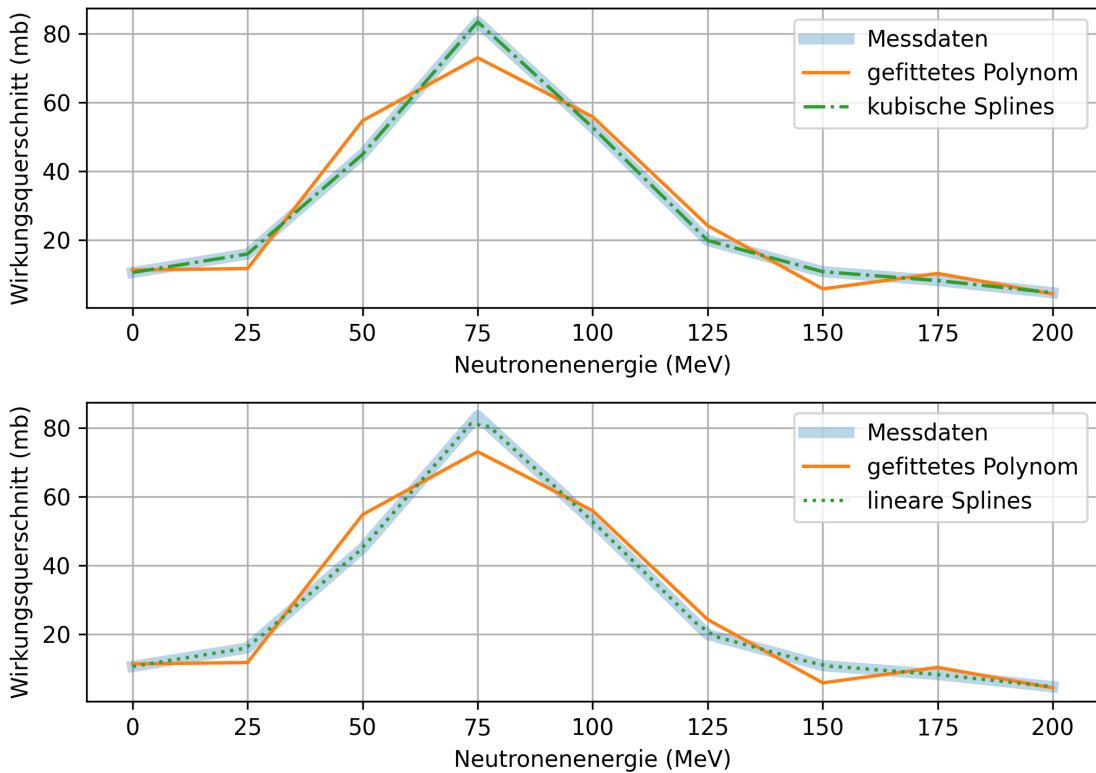
# Vergleich mit kubischen Splines
plt.subplot(2, 1, 1)
plt.plot(neutronenenergie, wirkungsquerschnitt, linewidth = 5, alpha=0.3, label="Messdaten")
plt.plot(neutronenenergie, fit, label = "gefittetes Polynom")
plt.plot(neutronenenergie, y_splines3, linestyle = 'dashdot', label = "kubische Splines")
plt.xlabel("Neutronenenergie (MeV)")
plt.ylabel("Wirkungsquerschnitt (mb)")
plt.legend()
plt.grid(True)

# Vergleich mit linearen Splines
plt.subplot(2, 1, 2)
plt.plot(neutronenenergie, wirkungsquerschnitt, linewidth = 5, alpha=0.3, label="Messdaten")
plt.plot(neutronenenergie, fit, label = "gefittetes Polynom")

plt.plot(x_neu, y_splines1, linestyle = 'dotted', label = 'lineare Splines')
plt.xlabel("Neutronenenergie (MeV)")
plt.ylabel("Wirkungsquerschnitt (mb)")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

```



5.: Die Splines bilden die Daten exakt nach. Das kann ein Vorteil sein, wenn keine Glättung der Daten gewünscht ist. Das Polynom ist dagegen nicht exakt an die Daten angepasst. Dafür können alle Datenpunkte mit einer einzigen Modellgleichung approximiert werden.

# **Part X**

## **a-energiedatenanalyse**

# Methodenbaustein Sensordatenanalyse

Bausteine Computergestützter Datenanalyse von Lukas Arnold, Simone Arnold, Florian Bagemihl, Matthias Baitsch, Marc Fehr, Franca Hollmann,



Maik Poetzsch und Sebastian Seipel. Methodenbaustein Sensordatenanalyse von Maik Poetzsch ist lizenziert unter [CC BY 4.0](#). Das Werk ist abrufbar auf [GitHub](#). Ausgenommen von der Lizenz sind alle Logos Dritter und anders gekennzeichneten Inhalte. 2025

<https://github.com/bausteine-der-datenanalyse/m-sensordatenanalyse>

# Voraussetzungen

## **Lernziele**

- 
- 
- 
- 
- 
-

# 65 Das Prinzip von Messungen

“In der Physik existiert nur das, was gemessen worden ist” (Merz 1968, 14).  
Merz, Ludwig. 1968. “Grundkurs der Messtechnik. Teil I: Das Messen elektrischer Größen.” 2. Auflage. München; Wien. R. Oldenbourg Verlag.

```
import numpy as np
import numpy.polynomial.polynomial as poly
import pandas as pd
import matplotlib.pyplot as plt
import scipy
import glob
```

- • • •

## Menschen im Tagesverlauf



- (a) Gerade Messabschnitte von 200 km Länge, Gesamtlänge ungefähr 2350 km
- (b) Gerade Messabschnitte von 100 km Länge, Gesamtlänge ungefähr 2775 km
- (c) Gerade Messabschnitte von 50 km Länge, Gesamtlänge ungefähr 3425 km

Figure 65.1: Küstenlinienparadox

## 65.1 Messung

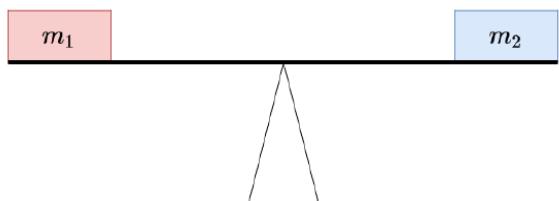
**!** Important ??: Messung

“Eine Messung ist der experimentelle Vorgang, durch den ein spezieller Wert einer physikalischen Größe als Vielfaches einer Einheit oder eines Bezugswertes ermittelt wird. Die Messung ergibt zunächst einen Messwert. Dieser stimmt aber aufgrund störender Einflüsse mit dem wahren Wert der Messgröße praktisch nie überein, sondern weist eine gewisse Messabweichung auf. Zum *vollständigen Messergebnis* wird der Messwert, wenn er mit quantitativen Aussagen über die zu erwartende Größe der Messabweichung ergänzt wird. Dies wird in der Messtechnik als Teil der Messaufgabe und damit der Messung verstanden.”

Messung, von verschiedenen [Autor:innen](#) steht unter der Lizenz [CC BY-SA 4.0](#) ist abrufbar auf [Wikipedia](#). 2025

- Die ideale Messung ist eine direkte Messung oder der gesuchte Wert hängt linear vom gemessenen Wert ab.
- Die ideale Messung ist *genau* und *präzise*.

### 65.1.1 Direkte und indirekte Messung



(a) Balkenwaage



(b) Zollstock

Figure 65.2: Direkte Messung



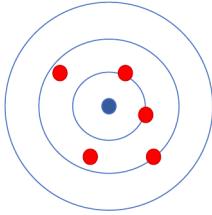
(a) Federwaage



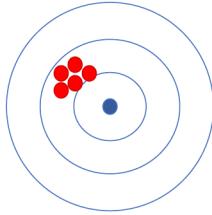
(b) Laserentfernungsmessung

Figure 65.3: Indirekte Messung

### 65.1.2 Genauigkeit und Präzision



(a) Genauigkeit



(b) Präzision

Die Genauigkeit einer Messung ist ein Maß für die Abweichung der Messwerte vom realen Wert. Die Präzision einer Messung beschreibt, wie gut die einzelnen Messwerte miteinander übereinstimmen. Die Genauigkeit ist nur bestimmt, wenn anerkannte Referenzwerte vorliegen, über die Standardabweichung der Stichprobe bestimmt.

Figure 65.4: Genauigkeit und Präzision

### 65.2 Messreihen

### 65.3 Varianz

## 65.4 Standardabweichung

### ⚠ Warning ??: Standardabweichung und Varianz in der Grundgesamtheit

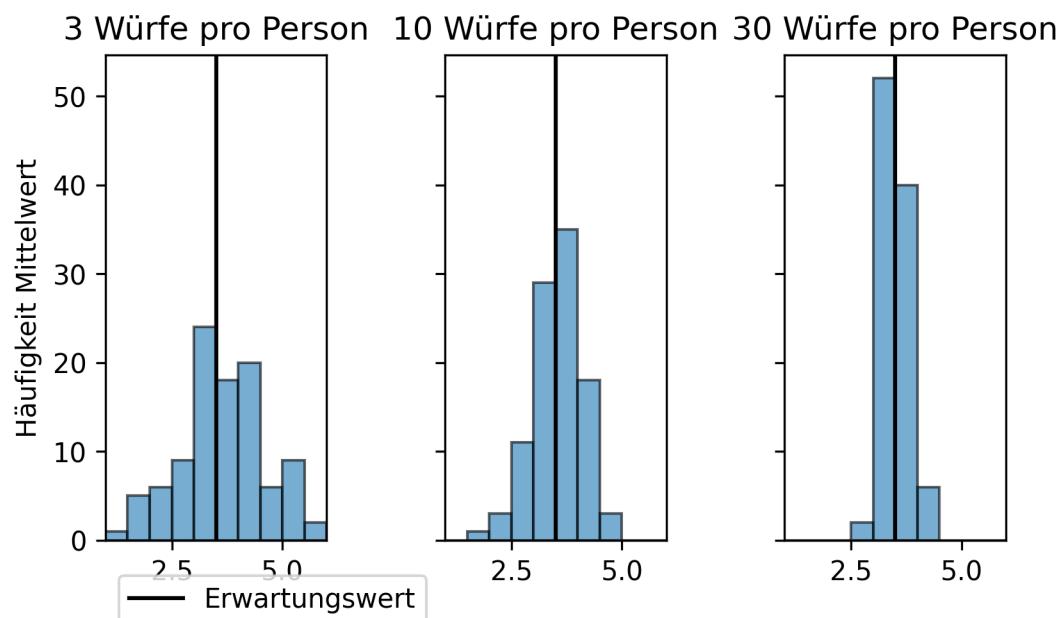
In der Stochastik werden Formeln häufig auch mit griechischen Buchstaben geschrieben, wenn Sie sich statt auf eine Stichprobe auf die Grundgesamtheit beziehen. Der Mittelwert in der Grundgesamtheit wird auch Erwartungswert genannt und mit dem griechischen Buchstaben  $\mu$  (My) dargestellt. Die Standardabweichung des Erwartungswerts wird mit  $\sigma$  (Sigma) gekennzeichnet.

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

#### **65.4.1 Experiment Verteilungskenngrößen**

## 65.5 Ergebnisse

## 65.6 grafische Darstellung



## 65.7 Code

```
personen = 100
standardabweichung_grundgesamtheit = np.arange(1, 7).std(ddof = 0)
seed = 1

# 3 Würfe
würfe = 3

## Personen stehen in den Zeilen (axis = 0), Würfe in den Spalten (axis = 1)
augen3 = np.random.default_rng(seed = seed).integers(low = 1, high = 6, endpoint = True, size = (personen, würfe))

## zeilenweise Mittelwert bilden mit np.array.mean(axis = 1)
print(f"Würfe pro Person: {würfe}\t\t\t\t",
      f"Stichprobengröße: {würfe * personen}\n",
      f"kleinster Mittelwert: {augen3.mean(axis = 1).min():.2f}\t\t\t",
      f"größter Mittelwert: {augen3.mean(axis = 1).max():.2f}\n",
      f"Stichprobenmittelwert: {augen3.mean():.2f}\t\t\t",
      f"Standardfehler: {standardabweichung_grundgesamtheit / ( augen3.size ** (1/2) ):.2f}\n",
      sep = "")

# 10 Würfe
würfe = 10

## Personen stehen in den Zeilen (axis = 0), Würfe in den Spalten (axis = 1)
augen10 = np.random.default_rng(seed = seed).integers(low = 1, high = 6, endpoint = True, size = (personen, würfe))

## zeilenweise Mittelwert bilden mit np.array.mean(axis = 1)
print(f"Würfe pro Person: {würfe}\t\t\t\t",
      f"Stichprobengröße: {würfe * personen}\n",
      f"kleinster Mittelwert: {augen10.mean(axis = 1).min():.2f}\t\t\t",
      f"größter Mittelwert: {augen10.mean(axis = 1).max():.2f}\n",
      f"Stichprobenmittelwert: {augen10.mean():.2f}\t\t\t",
      f"Standardfehler: {standardabweichung_grundgesamtheit / ( augen10.size ** (1/2) ):.2f}\n",
      sep = "")

# 50 Würfe
würfe = 50

## Personen stehen in den Zeilen (axis = 1), Würfe in den Spalten (axis = 1)
```

```

augen50 = np.random.default_rng(seed = seed).integers(low = 1, high = 6, endpoint = True, size = 50)

## zeilenweise Mittelwert bilden mit np.array.mean(axis = 1)
print(f"Würfe pro Person: {würfe}\t\t\t",
      f"Stichprobengröße: {würfe * personen}\n",
      f"kleinster Mittelwert: {augen50.mean(axis = 1).min():.2f}\t\t",
      f"größter Mittelwert: {augen50.mean(axis = 1).max():.2f}\n",
      f"Stichprobenmittelwert: {augen50.mean():.2f}\t\t",
      f"Standardfehler: {standardabweichung_grundgesamtheit / ( augen50.size ** (1/2) ):.2f}\n",
      sep = "")

```

```

personen = 100
standardabweichung_grundgesamtheit = np.arange(1, 7).std(ddof = 0)
seed = 1

# 3 Würfe
würfe = 3
augen3 = np.random.default_rng(seed = seed).integers(low = 1, high = 6, endpoint = True, size = 3)

# 10 Würfe
würfe = 10
augen10 = np.random.default_rng(seed = seed).integers(low = 1, high = 6, endpoint = True, size = 10)

# 50 Würfe
würfe = 50
augen50 = np.random.default_rng(seed = seed).integers(low = 1, high = 6, endpoint = True, size = 50)

# plotten
bins = 10

# 3 Würfe
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, sharey = True)

ax1.hist(augen3.mean(axis = 1), bins = bins, alpha = 0.6, edgecolor = 'black', range = (1, 6))
ax1.set_xlim(1, 6)
ax1.axvline(x = 3.5, ymin = 0, ymax = 1, color = 'black', label = 'Erwartungswert')
ax1.set_ylabel('mittleres Würfelergebnis')
ax1.set_ylabel('Häufigkeit Mittelwert')
ax1.set_title("3 Würfe pro Person")
ax1.legend(loc = 'lower left', bbox_to_anchor = (0, -0.2))

```

```

# 10 Würfe
ax2.hist(augen10.mean(axis = 1), bins = bins, alpha = 0.6, edgecolor = 'black', range = (1, 6))
ax2.set_xlim(1, 6)
ax2.axvline(x = 3.5, ymin = 0, ymax = 1, color = 'black')
ax2.set_ylabel('mittleres Würfelergebnis')
ax2.set_title("10 Würfe pro Person")

# 30 Würfe
ax3.hist(augen50.mean(axis = 1), bins = bins, alpha = 0.6, edgecolor = 'black', range = (1, 6))
ax3.set_xlim(1, 6)
ax3.axvline(x = 3.5, ymin = 0, ymax = 1, color = 'black')
ax3.set_ylabel('mittleres Würfelergebnis')
ax3.set_title("30 Würfe pro Person")

plt.tight_layout()
plt.show()

```

### 65.7.1 Aufgabe Verteilungskenngrößen

**Listing 65.1**

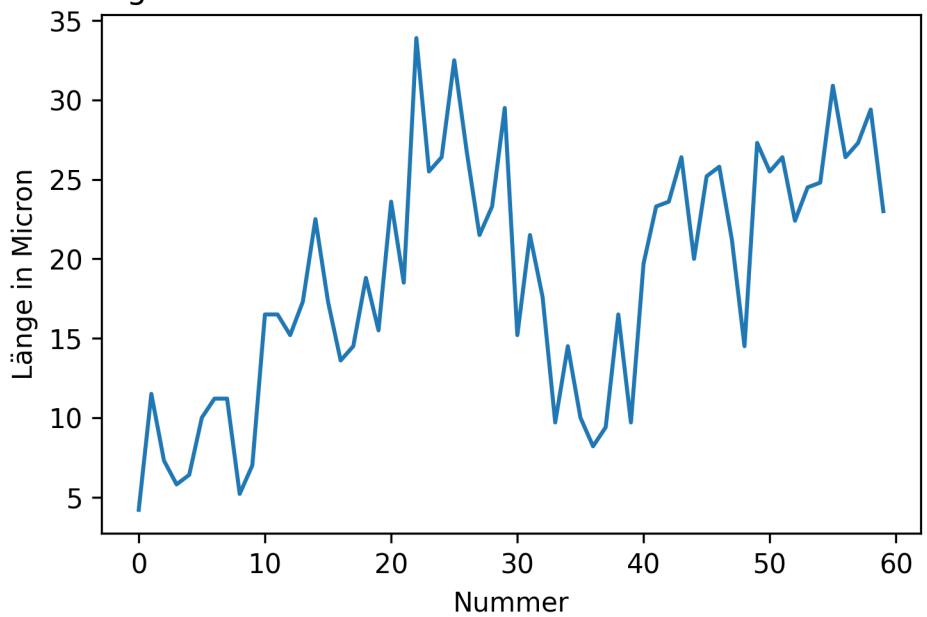
```

dateipfad = "01-daten/ToothGrowth.csv"
meerschweinchen = pd.read_csv(filepath_or_buffer = dateipfad, sep = ',', header = 0, \
                               names = ['ID', 'len', 'supp', 'dose'], dtype = {'ID': 'int', 'len': 'float', 'dose': 'float'})

```

<https://doi.org/10.1093/jn/33.5.491>

Länge der zahnbildenden Zellen von Meerschweinchen



💡 Tip ??: Musterlösung Verteilungskenngrößen

```
def Verteilungskennwerte(x, output = True):

    # Anzahl Messwerte bestimmen
    N = len(x)

    # arithmetisches Mittel bestimmen
    stichprobenmittelwert = sum(x) / N

    # Stichprobenvarianz bestimmen
    stichprobenvarianz = sum((x - stichprobenmittelwert) ** 2) / (N - 1)

    # Standardabweichung bestimmen
    standardabweichung = stichprobenvarianz ** (1/2)

    # Stichprobenfehler bestimmen
    stichprobenfehler = standardabweichung / (N ** (1/2))

    # Ausgabe
    if output: # output = True
        print(f"N: {N}\n",
              f"arithmetisches Mittel: {stichprobenmittelwert:.2f}\n",
              f"Stichprobenfehler: {stichprobenfehler:.2f}\n",
              f"Stichprobenvarianz: {stichprobenvarianz:.2f}\n",
              f"Standardabweichung: {standardabweichung:.2f}",
              sep = ' ')
    else: # output = False
        return N, stichprobenmittelwert, stichprobenfehler, stichprobenvarianz, standardabweichung

Verteilungskennwerte(meerschweinchen['len'])
```

### 65.7.2 Varianz und Standardabweichung mit NumPy und Pandas

```
print("Varianz:")
print(f"NumPy:\t{np.var(meerschweinchen['len']):.2f}")
print(f"Pandas:\t{meerschweinchen['len'].var():.2f}")

print("\nStandardabweichung:")
print(f"NumPy:\t{np.std(meerschweinchen['len']):.2f}")
print(f"Pandas:\t{meerschweinchen['len'].std():.2f}")
```

# 66 Die Normalverteilung

zentrale Grenzwertsatz

**i** Note ??: Häufigkeitsverteilung von Würfelergebnissen

Für einen Würfel gibt es 6 mögliche Ergebnisse, für 2 Würfel  $6 * 6$  mögliche Kombinationen, für 3 Würfel  $6 * 6 * 6$  Kombinationen und so weiter. Weil viele Kombinationen wertgleich sind, kommen Wurfergebnisse in der Nähe des Erwartungswerts häufiger vor als beispielsweise ein Einserpasch.

## 66.1 ein Würfel

```
ein_würfel = []

for i in range(1, 7):
    ein_würfel.append(i)

ein_würfel = pd.Series(ein_würfel)

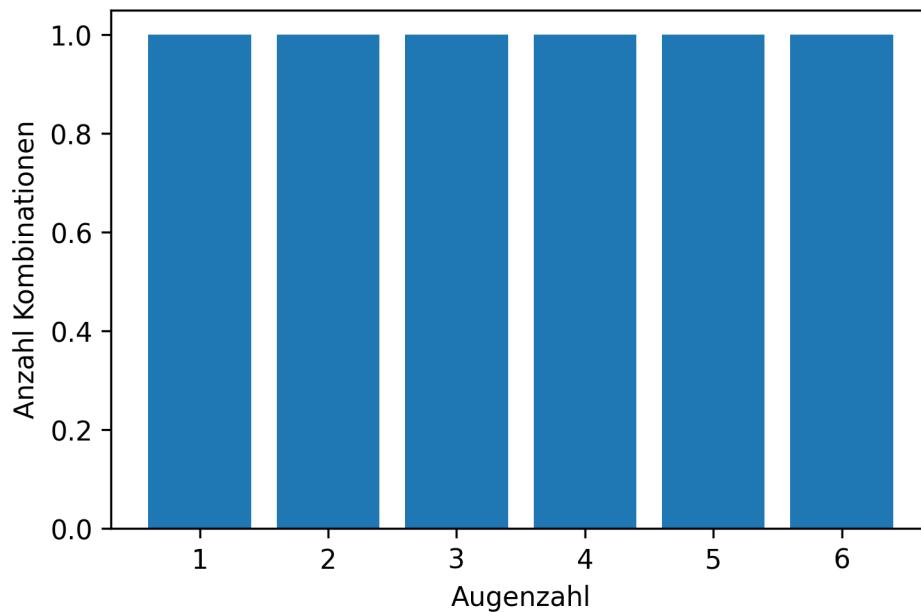
print("Häufigkeitsverteilung der Augensumme:")
print(ein_würfel.value_counts(), "\n")
print(f"Durchschnitt: {ein_würfel.mean():.1f}")

plt.bar(ein_würfel.unique(), ein_würfel.value_counts())
plt.xlabel('Augenzahl')
plt.ylabel('Anzahl Kombinationen')
plt.show()

Häufigkeitsverteilung der Augensumme:
1    1
2    1
```

```
3    1  
4    1  
5    1  
6    1  
Name: count, dtype: int64
```

Durchschnitt: 3.5



## 66.2 zwei Würfel

```

zwei_würfel = []

for i in range(1, 7):
    würfel_1 = i

    for j in range (1, 7):
        würfel_2 = j
        zwei_würfel.append(würfel_1 + würfel_2)

zwei_würfel = pd.Series(zwei_würfel)

print("Häufigkeitsverteilung der Augensumme:")
print(zwei_würfel.value_counts().sort_index(ascending = True), "\n")
print(f"Durchschnitt: {zwei_würfel.mean():.1f}")
print(f"Durchschnitt pro Würfel: {zwei_würfel.mean() / 2:.1f}")

plt.bar(zwei_würfel.unique(), zwei_würfel.value_counts().sort_index(ascending = True))
plt.xlabel('Augenzahl')
plt.ylabel('Anzahl Kombinationen')
plt.grid()
plt.show()

```

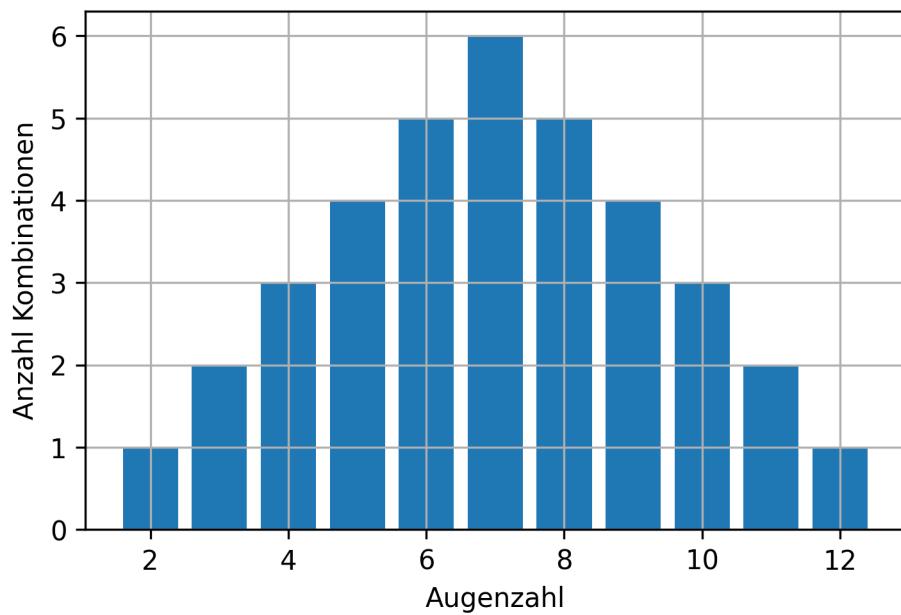
Häufigkeitsverteilung der Augensumme:

2	1
3	2
4	3
5	4
6	5
7	6
8	5
9	4
10	3
11	2
12	1

Name: count, dtype: int64

Durchschnitt: 7.0

Durchschnitt pro Würfel: 3.5



### 66.3 drei Würfel

```

drei_würfel = []

for i in range(1, 7):
    würfel_1 = i

    for j in range (1, 7):
        würfel_2 = j

        for k in range (1, 7):
            würfel_3 = k
            drei_würfel.append(würfel_1 + würfel_2 + würfel_3)

drei_würfel = pd.Series(drei_würfel)

print("Häufigkeitsverteilung der Augensumme:")
print(drei_würfel.value_counts().sort_index(ascending = True), "\n")
print(f"Durchschnitt: {drei_würfel.mean():.1f}")
print(f"Durchschnitt pro Würfel: {drei_würfel.mean() / 3:.1f}")

plt.bar(drei_würfel.unique(), drei_würfel.value_counts().sort_index(ascending = True))
plt.xlabel('Augenzahl')
plt.ylabel ('Anzahl Kombinationen')
plt.grid()
plt.show()

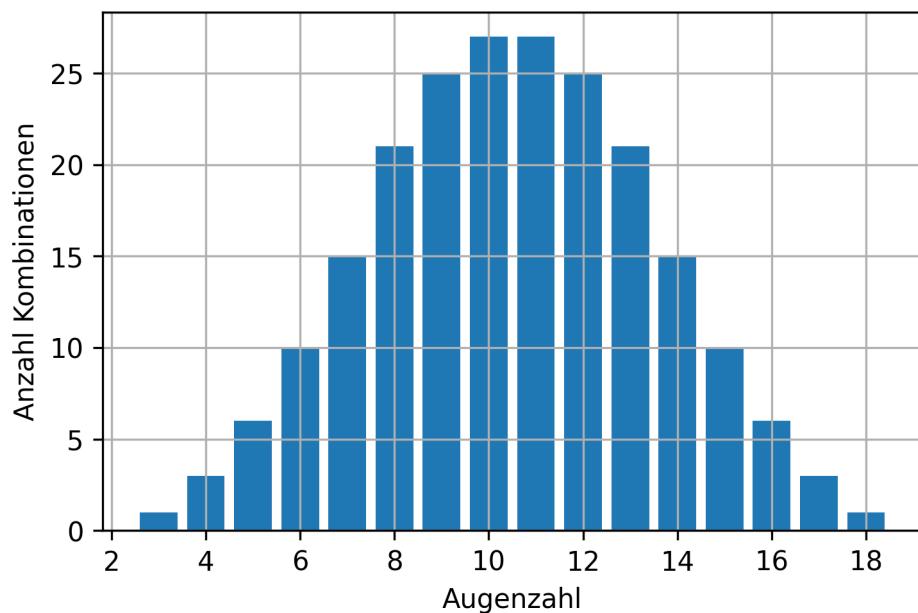
```

Häufigkeitsverteilung der Augensumme:

3	1
4	3
5	6
6	10
7	15
8	21
9	25
10	27
11	27
12	25
13	21
14	15
15	10
16	6
17	3

```
18      1  
Name: count, dtype: int64
```

Durchschnitt: 10.5  
Durchschnitt pro Würfel: 3.5



Normalverteilung

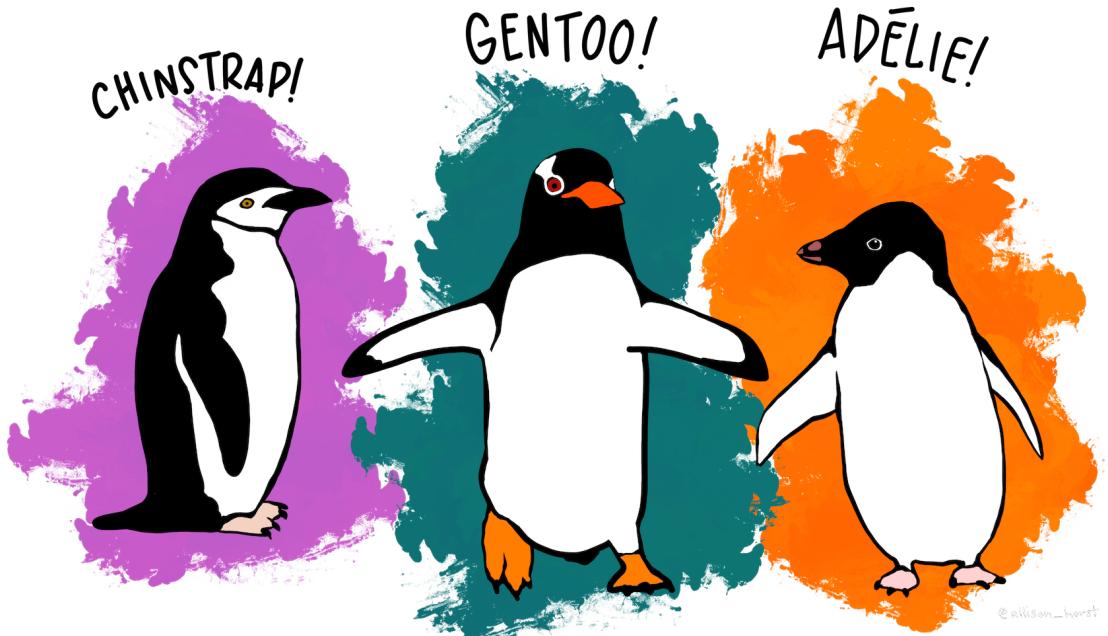


Figure 66.1: Pinguine des Palmer-Station-Datensatzes

CC0-1.0

[GitHub](#)

CCO

[GitHub](#)

```
# R Befehle, um den Datensatz zu laden
install.packages("palmerpenguins")
library(palmerpenguins)
```

<https://allisonhorst.github.io/palmerpenguins/>

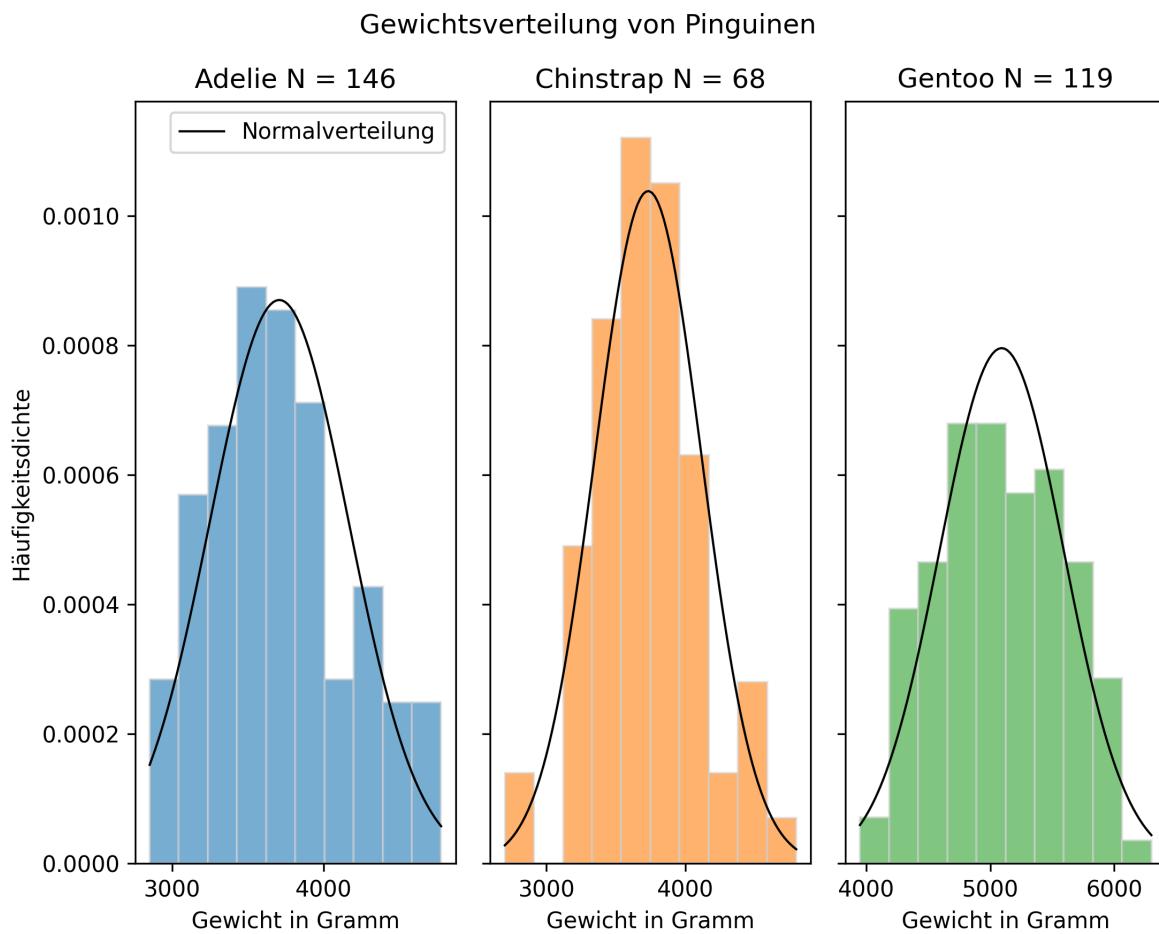
```
penguins = pd.read_csv(filepath_or_buffer = "01-daten/penguins.csv")

# Tiere mit unvollständigen Einträgen entfernen
penguins.drop(np.where(penguins.apply(pd.isna).any(axis = 1))[0], inplace = True)

print(penguins.info(), "\n");
```

```
print(penguins.groupby(by = penguins['species']).size())
```

## 66.4 Grafik



## 66.5 Code

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (7.5, 6), sharey = True, layout = 'tight')
plt.suptitle('Gewichtsverteilung von Pinguinen')

# Adelie
species = 'Adelie'
data = penguins['body_mass_g'][penguins['species'] == species]

## Histogramm
ax1.hist(data, alpha = 0.6, edgecolor = 'lightgrey', color = 'C0', density = True)
```

```

ax1.set_xlabel('Gewicht in Gramm')
ax1.set_ylabel('Häufigkeitsdichte')
ax1.set_title(label = str(species) + " N = " + str(data.size))

## Normalverteilungskurve
stichprobenmittelwert = data.mean()
stichprobenstandardabweichung = data.std(ddof = 1)
hist, bin_edges = np.histogram(data)
x_values = np.linspace(min(bin_edges), max(bin_edges), 100)
y_values = 1 / (stichprobenstandardabweichung * np.sqrt(2 * np.pi)) * np.exp(- (x_values - s

ax1.plot(x_values, y_values, color = 'black', linewidth = 1, label = 'Normalverteilung')
ax1.legend()

# Chinstrap
species = 'Chinstrap'
data = penguins['body_mass_g'][penguins['species'] == species]

## Histogramm
ax2.hist(data, alpha = 0.6, edgecolor = 'lightgrey', color = 'C1', density = True)
ax2.set_xlabel('Gewicht in Gramm')
ax2.set_title(label = str(species) + " N = " + str(data.size))

## Normalverteilungskurve
stichprobenmittelwert = data.mean()
stichprobenstandardabweichung = data.std(ddof = 1)
hist, bin_edges = np.histogram(data)
x_values = np.linspace(min(bin_edges), max(bin_edges), 100)
y_values = 1 / (stichprobenstandardabweichung * np.sqrt(2 * np.pi)) * np.exp(- (x_values - s

ax2.plot(x_values, y_values, color = 'black', linewidth = 1)

# Gentoo
species = 'Gentoo'
data = penguins['body_mass_g'][penguins['species'] == species]

## Histogramm
ax3.hist(data, alpha = 0.6, edgecolor = 'lightgrey', color = 'C2', density = True)
ax3.set_xlabel('Gewicht in Gramm')
ax3.set_title(label = str(species) + " N = " + str(data.size))

## Normalverteilungskurve

```

```

stichprobenmittelwert = data.mean()
stichprobenstandardabweichung = data.std(ddof = 1)
hist, bin_edges = np.histogram(data)
x_values = np.linspace(min(bin_edges), max(bin_edges), 100)
y_values = 1 / (stichprobenstandardabweichung * np.sqrt(2 * np.pi)) * np.exp(- (x_values - stichprobenmittelwert) ** 2 / (2 * stichprobenstandardabweichung ** 2))

ax3.plot(x_values, y_values, color = 'black', linewidth = 1)

plt.show()

```

### ! Important ??: Histogramm

Das Histogramm ist eine grafische Darstellung der Häufigkeitsverteilung kardinal skaliertes Merkmale (d. h. mit numerischen, geordneten Merkmalsausprägungen). Die Daten werden in Klassen, die eine konstante oder variable Breite haben können, eingeteilt. Es werden direkt nebeneinanderliegende Rechtecke von der Breite der jeweiligen Klasse gezeichnet, deren Flächeninhalte die (relativen oder absoluten) Klassenhäufigkeiten darstellen. Die Höhe jedes Rechtecks stellt dann die (relative oder absolute) Häufigkeitsdichte dar, also die (relative oder absolute) Häufigkeit dividiert durch die Breite der entsprechenden Klasse.

#### Note 66.2: Histogramm berechnen und visualisieren

Als Beispiel wird die Länge der zahnbildenden Zellen der Meerschweinchen verwendet, die eine Vitamin-C-Dosis von 2 erhielten.

```

dose2 = meerschweinchen.loc[meerschweinchen['dose'] == 2, 'len']
print(*list(dose2)) # * = Ausgabe ohne Kommata
print("N", len(dose2), "Minimum:", dose2.min(), "Maximum:", dose2.max(), "Spannweite", dose2.max() - dose2.min())

```

23.6 18.5 33.9 25.5 26.4 32.5 26.7 21.5 23.3 29.5 25.5 26.4 22.4 24.5 24.8 30.9 26.4 28.5  
N 20 Minimum: 18.5 Maximum: 33.9 Spannweite 15.399999999999999

Mit der Funktion `np.histogram(a, bins = 10, range = None, density = None)` kann ein Histogramm berechnet werden.

- `a` sind die zu berechnenden Daten
- `bins` spezifiziert die Anzahl an Klassen, standardmäßig werden 10 gewählt.

- `range = (float, float)` erlaubt es, die untere und obere Grenze der Klassen festzulegen.
- `density = True` erlaubt es statt der absoluten Häufigkeiten, den Wert der Häufigkeitsdichtefunktion darzustellen. Dies berechnet sich wie folgt:
  - relative Häufigkeit = Anzahl Werte je Klasse / Anzahl aller Werte
  - Häufigkeitsdichte = Anzahl Werte je Klasse / (Anzahl aller Werte \* Klassenbreite)
  - Klassenbreite = Maximum(Werte) - Minimum(Werte) / Anzahl Klassen

Für die überschaubare Anzahl an Werten wird ein Histogramm mit 5 Klassen berechnet. Zum Vergleich wird auch die Häufigkeitsdichte ausgegeben.

```
print(np.histogram(dose2, bins = 5))
print("Häufigkeitsdichte:", np.histogram(dose2, bins = 5, density = True)[0])

(array([2, 5, 8, 2, 3]), array([18.5 , 21.58, 24.66, 27.74, 30.82, 33.9 ]))
Häufigkeitsdichte: [0.03246753 0.08116883 0.12987013 0.03246753 0.0487013 ]
```

Die Funktion `np.histogram()` gibt an erster Stelle ein array mit den absoluten Häufigkeiten bzw. der Häufigkeitsdichte jeder Klasse zurück. An zweiter Stelle wird ein array mit den x-Positionen der Klassenrechtecke zurückgegeben - dabei wird für jede Klasse die Position der linken Seite sowie für die letzte Klasse zusätzlich die Position der rechten Seite des Rechtecks ausgegeben. Für 5 Klassen werden also 6 Positions-werte ausgegeben.

Die Klassenbreite kann zum Beispiel mit der Methode `np.diff()` ausgegeben werden.

```
hist_abs, bin_edges = np.histogram(dose2, bins = 5)
klassenbreite = np.diff(bin_edges)
print(klassenbreite)
```

[3.08 3.08 3.08 3.08 3.08]

Durch Multiplikation der Häufigkeitsdichte mit der Klassenbreite können die relativen Häufigkeiten berechnet werden.

```
hist_dichte = np.histogram(dose2, bins = 5, density = True)[0]
hist_relativ = hist_dichte * klassenbreite
print(hist_relativ)
```

```
[0.1  0.25 0.4  0.1  0.15]
```

Die Summe der relativen Häufigkeiten ist 1.

Ein Histogramm kann mit der Funktion `plt.hist(x, bins = None, *, range = None, density = False)` aufgerufen werden, welche intern `np.histogram()` für die Berechnungen aufruft. Die Parameter der Funktion entsprechen denen der NumPy-Funktion, wobei mit dem Argument `x` die darzustellenden Daten übergeben werden. Zusätzlich können verschiedene Grafikparameter übergeben werden.

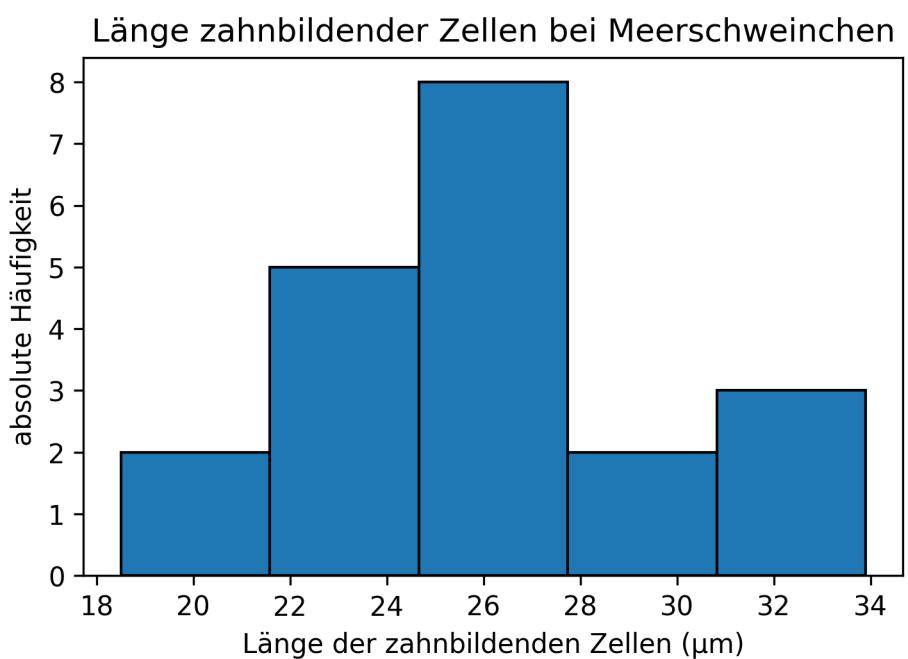
Die Funktion hat 3 Rückgabewerte: die absolute Häufigkeit der Klassen (bzw. wenn `density = True` die Häufigkeitsdichte), die x-Position der Rechtecke und die Objekte für die Grafikerstellung (letztere werden im folgenden Code im Objekt `ignore` gespeichert und nicht weiter verwendet.).

## 66.6 absolute Häufigkeit

```
plt.hist(dose2, bins = 5, edgecolor = 'black')
plt.title('Länge zahnbildender Zellen bei Meerschweinchen')

# Achsenbeschriftung
plt.xlabel('Länge der zahnbildenden Zellen (m)')
plt.ylabel('absolute Häufigkeit')

plt.show()
```



## 66.7 relative Häufigkeit

Eine Darstellung der relativen Häufigkeiten ist nicht direkt möglich.

```

hist_dichte, bins, ignore = plt.hist(dose2, bins = 5, density = True, edgecolor = 'black')
plt.title('Länge zahnbildender Zellen bei Meerschweinchen')

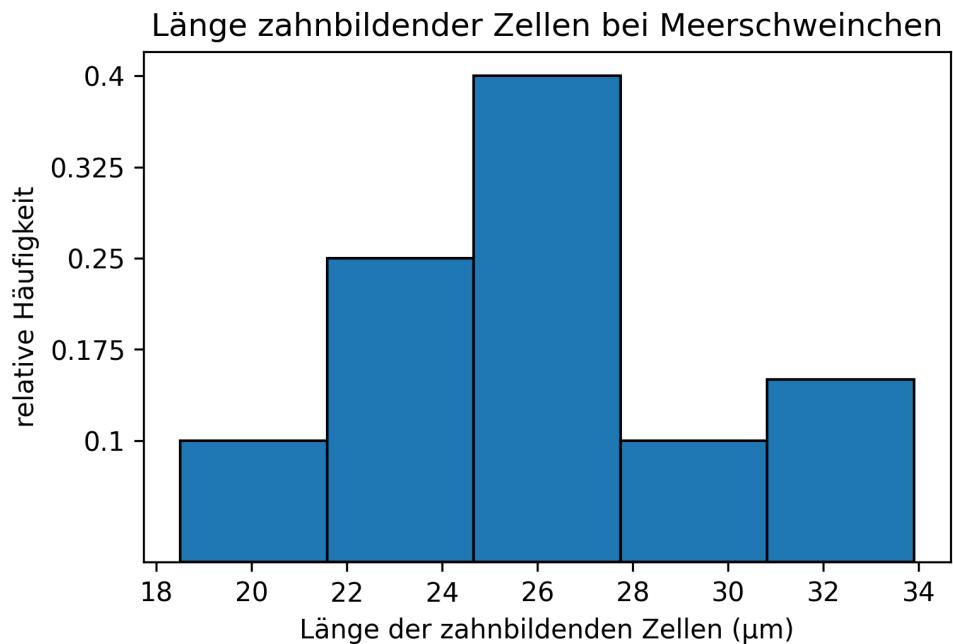
# relative Häufigkeit berechnen
klassenbreite = np.diff(bins)[0]
hist_relativ = hist_dichte * klassenbreite

# yticks erzeugen an der Position von min(hist_dichte) bis max(hist_dichte)
# aber mit Werten von hist_relativ
plt.yticks(ticks = np.linspace(min(hist_dichte), max(hist_dichte), len(hist_relativ)),
           labels = np.linspace(hist_relativ.round(2).min(), hist_relativ.round(2).max(), len(hist_relativ)))

# Achsenbeschriftung
plt.xlabel('Länge der zahnbildenden Zellen (μm)')
plt.ylabel('relative Häufigkeit')

plt.show()

```

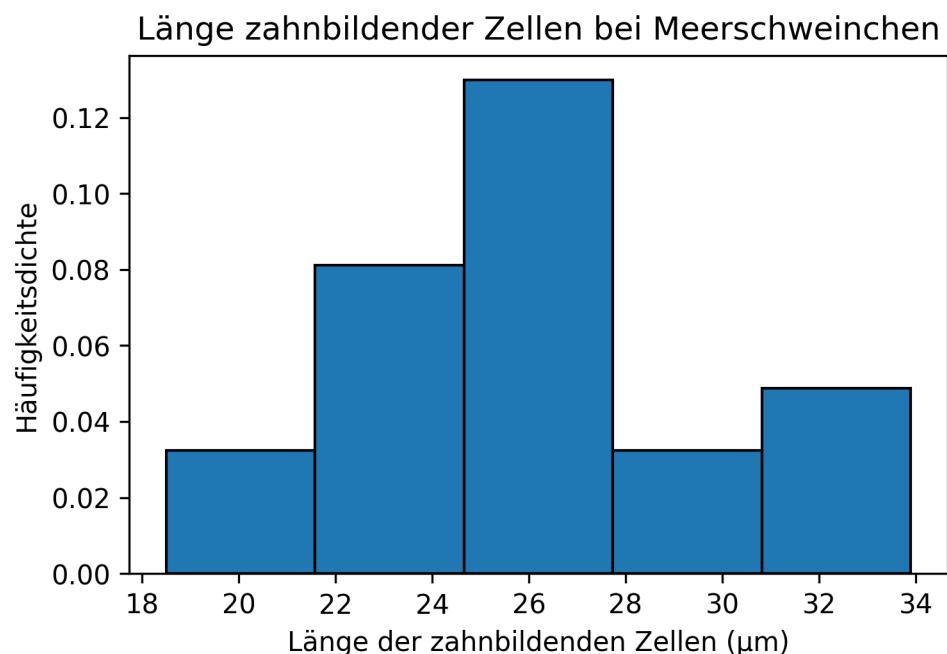


## 66.8 Häufigkeitsdichte

```
plt.hist(dose2, bins = 5, density = True, edgecolor = 'black')
plt.title('Länge zahnbildender Zellen bei Meerschweinchen')

# Achsenbeschriftung
plt.xlabel('Länge der zahnbildenden Zellen (m)')
plt.ylabel('Häufigkeitsdichte')

plt.show()
```



Histogramme sind nicht immer gut geeignet, um die Verteilung einer Stichprobe zu charakterisieren. Der visuelle Eindruck hängt von der gewählten Klassenzahl ab - ein Beispiel:

## 66.9 3 Klassen

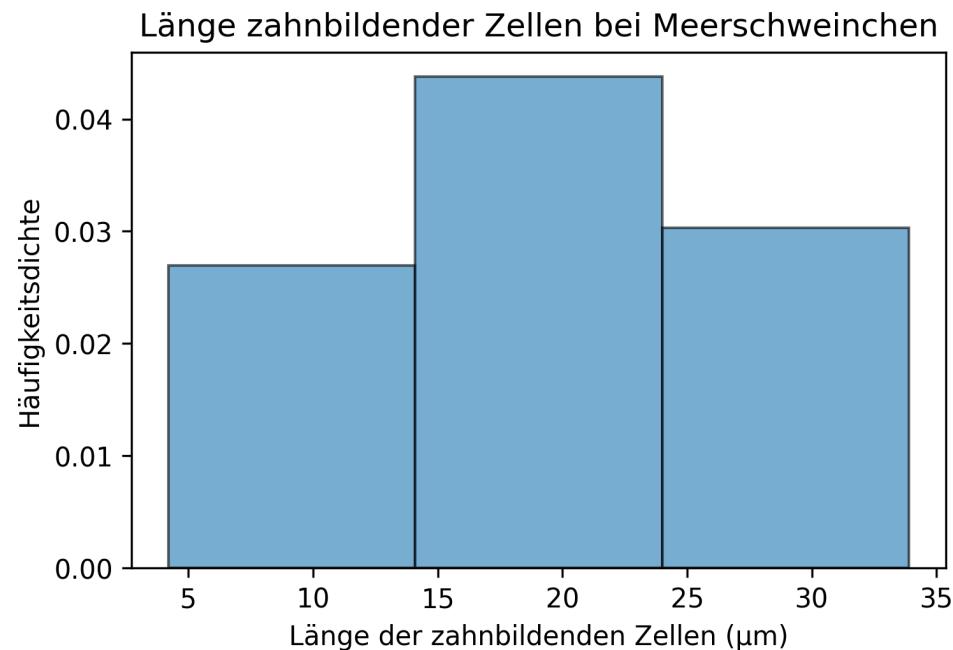
```

plt.hist(meerschweinchen['len'], bins = 3, density = True, edgecolor = 'black', alpha = 0.5)
plt.title('Länge zahnbildender Zellen bei Meerschweinchen')

# Achsenbeschriftung
plt.xlabel('Länge der zahnbildenden Zellen ( m )')
plt.ylabel('Häufigkeitsdichte')

plt.show()

```



## 66.10 5 Klassen

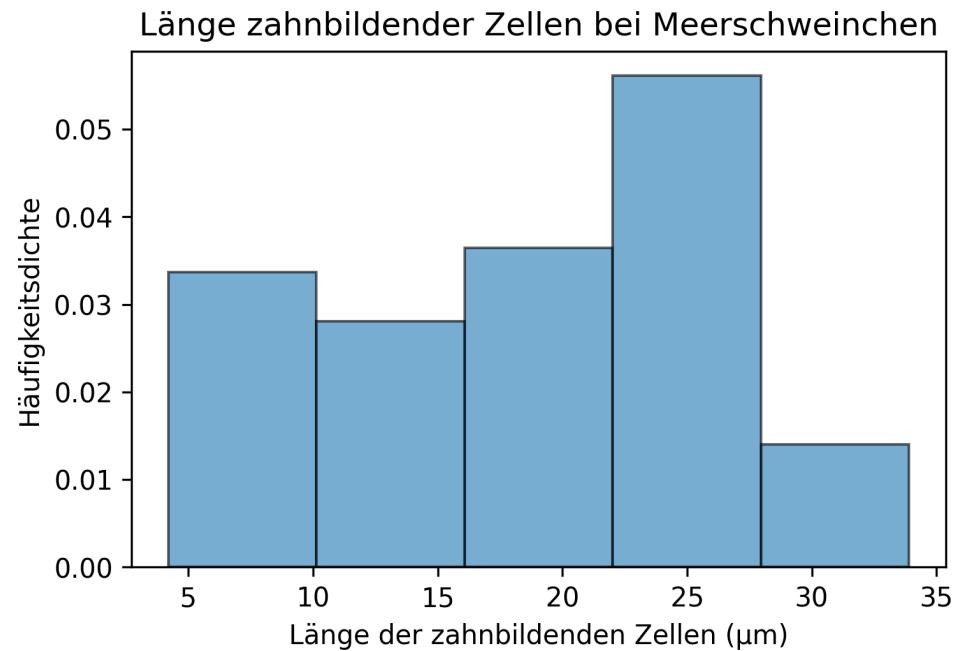
```

plt.hist(meerschweinchen['len'], bins = 5, density = True, edgecolor = 'black', alpha = 0.5)
plt.title('Länge zahnbildender Zellen bei Meerschweinchen')

# Achsenbeschriftung
plt.xlabel('Länge der zahnbildenden Zellen ( m )')
plt.ylabel('Häufigkeitsdichte')

plt.show()

```



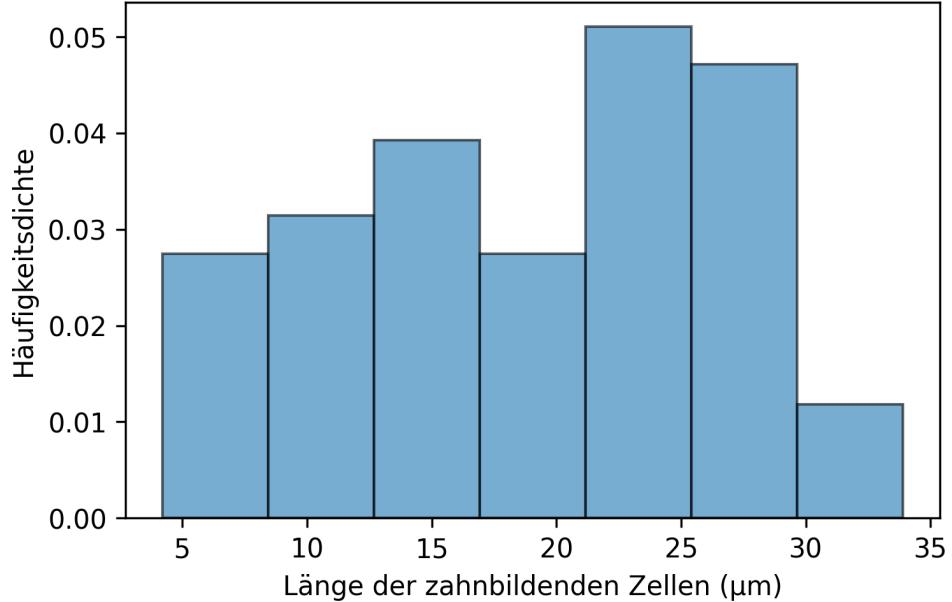
## 66.11 7 Klassen

```
plt.hist(meerschweinchen['len'], bins = 7, density = True, edgecolor = 'black', alpha = 0.8)
plt.title('Länge zahnbildender Zellen bei Meerschweinchen')

# Achsenbeschriftung
plt.xlabel('Länge der zahnbildenden Zellen ( m )')
plt.ylabel('Häufigkeitsdichte')

plt.show()
```

Länge zahnbildender Zellen bei Meerschweinchen



## 66.12 Normalverteilung anpassen

**i** Note ??: Dichtekurven berechnen und darstellen

Betrachten wir die Verteilungskennwerte der Gruppe der Meerschweinchen, die eine Dosis von 2 Milligramm Vitamin C erhielten.

```
print(verteilungskennwerte(dose2), "\n");

dose2_mean = verteilungskennwerte(dose2, output = False)[1]
dose2_std = verteilungskennwerte(dose2, output = False)[4]

print("Exakter Mittelwert:", dose2_mean)
print("Exakte Standardabweichung:", dose2_std)
```

```
N: 20
arithmetisches Mittel: 26.10
Stichprobenfehler: 0.84
Stichprobenvarianz: 14.24
Stichprobenstandardabweichung: 3.77
None
```

```
Exakter Mittelwert: 26.1
Exakte Standardabweichung: 3.7741503052098744
```

Wenn wir die Standardabweichung und das arithmetische Mittel in die Normalverteilungsfunktion einsetzen, erhalten wir:

$$f(x) = \frac{1}{3.7742\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-26.10}{3.7742})^2}$$
$$f(x) = 0.1057 \cdot e^{-\frac{1}{2}(\frac{x-26.10}{3.7742})^2}$$

In Python können die Berechnungen umgesetzt und grafisch dargestellt werden:

```

# Histogram der Häufigkeitsdichte zeichnen
plt.hist(dose2, bins = 7, density = True, edgecolor = 'black', alpha = 0.6);
plt.title('Länge zahnbildender Zellen bei Meerschweinchen')

# Achsenbeschriftung
plt.xlabel('Länge der zahnbildenden Zellen ( m )')
plt.ylabel('Häufigkeitsdichte')

# Normalverteilung berechnen.
hist, bin_edges = np.histogram(dose2, bins = 7)

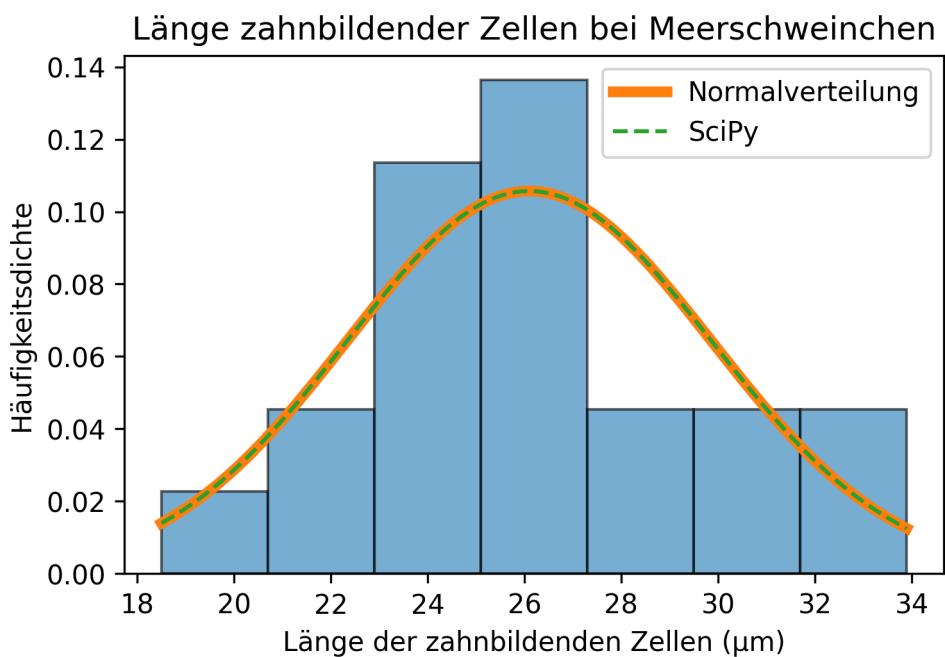
x_values = np.linspace(min(bin_edges), max(bin_edges), 100)

## Normalverteilungsfunktion mit Python berechnen
y_values = 1 / (dose2_std * np.sqrt(2 * np.pi)) * np.exp(- (x_values - dose2_mean) ** 2 /
plt.plot(x_values, y_values, label = 'Normalverteilung', lw = 4)

## scipy
y_values_scipy = scipy.stats.norm.pdf(x_values, loc = dose2_mean, scale = dose2_std)
plt.plot(x_values, y_values_scipy, label = 'SciPy', linestyle = 'dashed')

plt.legend()
plt.show()

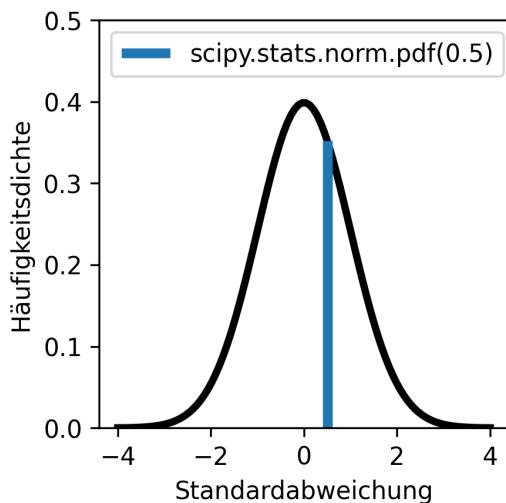
```



Die Verteilung der Länge zahnbildender Zellen bei Meerschweinchen, die eine Dosis von 2 Milligramm Vitamin C erhielten, könnte einer Normalverteilung entsprechen. Aufgrund der geringen Stichprobengröße ist dies aber schwer zu beurteilen.

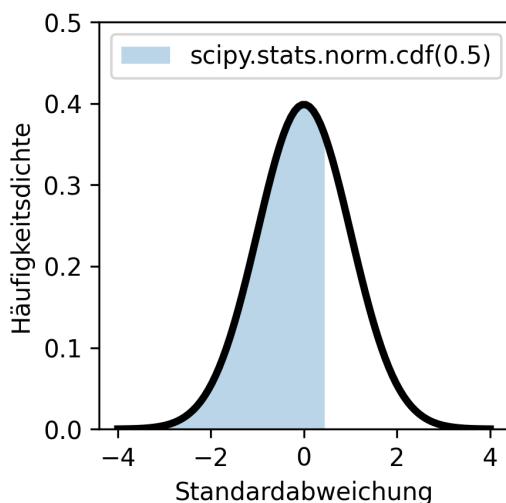
## 66.13 Das Paket SciPy

```
import scipy
print("Häufigkeitsdichte der Normalverteilung bei x = 0:", scipy.stats.norm.pdf(0), "\n")
```



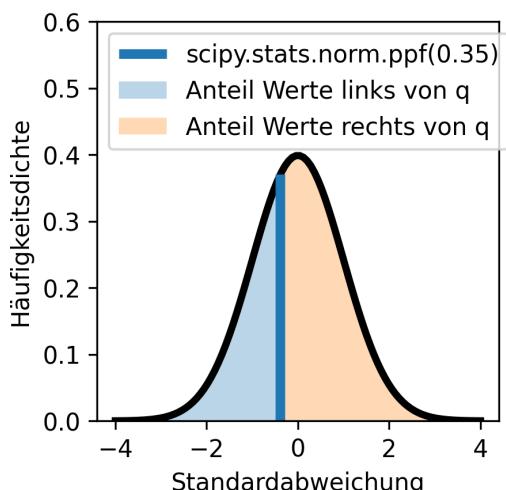
#### Beschreibung

Die Funktion `scipy.stats.norm.pdf(x)` berechnet die Dichte der Normalverteilung am Punkt  $x$  ( $\text{pdf} = \text{probability density function}$ ).  $x$  kann auch ein array sein - so wurde die linksstehende Kurve mit dem Befehl `scipy.stats.norm.pdf(np.linspace(-4, 4, 100))` berechnet.



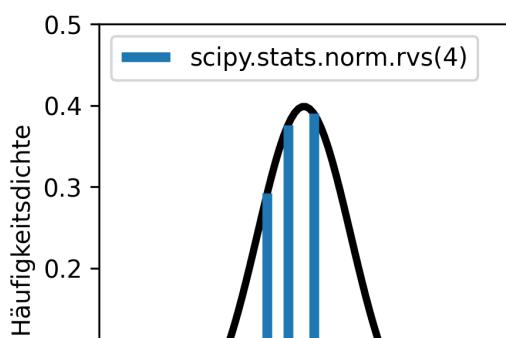
#### Beschreibung

Die Funktion `scipy.stats.norm.cdf(x)` berechnet den Anteil der Werte  $q$  links von  $x$  ( $\text{cdf} = \text{cumulative density function}$ ).



#### Beschreibung

Die Funktion `scipy.stats.norm.ppf(q)` ist die Quantilfunktion der Normalverteilung und die Umkehrfunktion der kumulativen Häufigkeitsdichtefunktion (cdf). Die Funktion berechnet für  $0 \leq q \leq 1$  den Wert  $x$ , links von dem der Anteil  $q$  aller Werte liegt und rechts von dem der Anteil  $1-q$  liegt ( $\text{ppf} = \text{percentile point function}$ ).



#### Beschreibung

Die Funktion `scipy.stats.norm.rvs(size)` zieht  $\text{size}$  Zufallszahlen aus der Normalverteilung.

*Hinweis:* Die Zufallszahlen werden im Skript dynamisch gezogen.

## 66.14 Aufgaben Normalverteilung

[Wikipedia](#)

1.

2.

- 
- 
- 

3.

im Land Niger

### 💡 Musterlösung Normalverteilung

Aufgabe 1: Einen IQ von mehr als ...

```
print(scipy.stats.norm.ppf(loc = 100, scale = 15, q = 0.98))
```

130.80623365947733

Aufgabe 2: Sie benötigen einen IQ von mindestens...

```
print(scipy.stats.norm.ppf(loc = 100, scale = 15, q = 0.99))
print(scipy.stats.norm.ppf(loc = 100, scale = 15, q = 0.999))
print(scipy.stats.norm.ppf(loc = 100, scale = 15, q = 1 - (0.003 / 100)))
```

134.8952181106126

146.3534845925172

160.19216216677682

Aufgabe 3: Nicht ganz.

```
print(scipy.stats.norm.cdf(loc = 71, scale = 15, x = 100))
```

```
0.9734024259789904
```

der Spiegel berichtet

## 66.15 Konfidenzintervalle

1.

- 
- 
- 

2.

3.

Fehler 1. und 2. Art

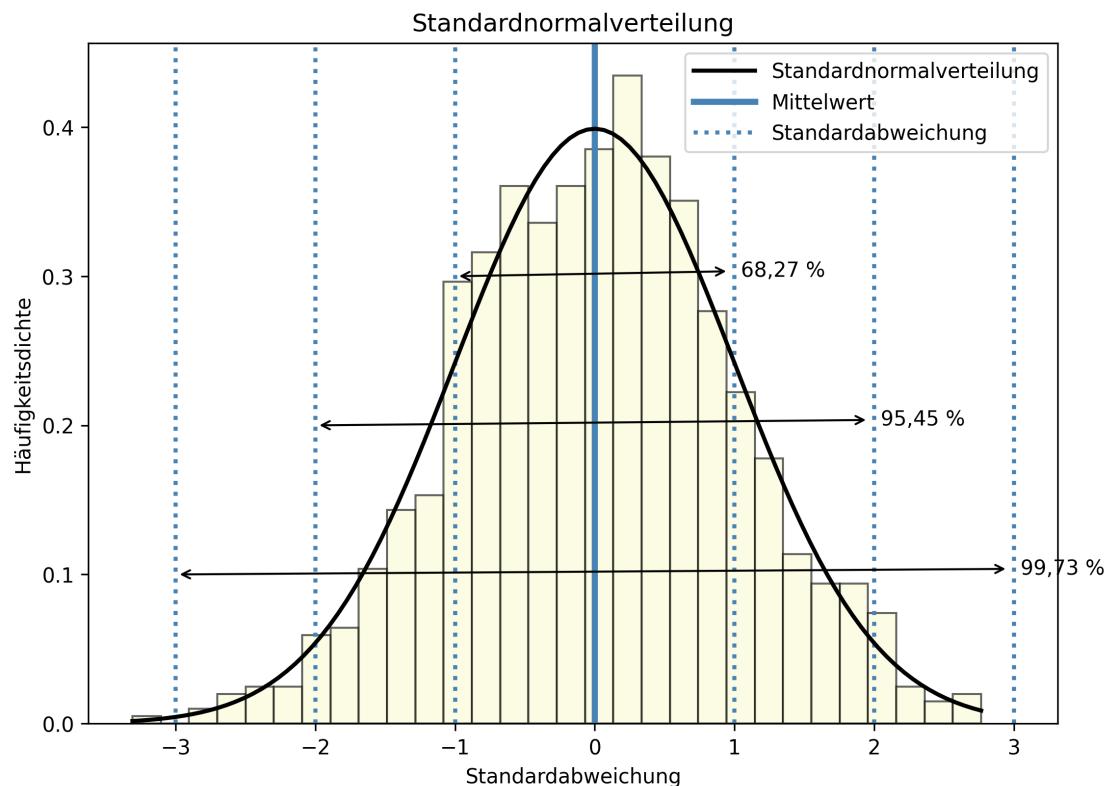
- 
- 
-

**i** Note ??: Prinzip der schließenden Statistik

## 66.16 Standardnormalverteilung

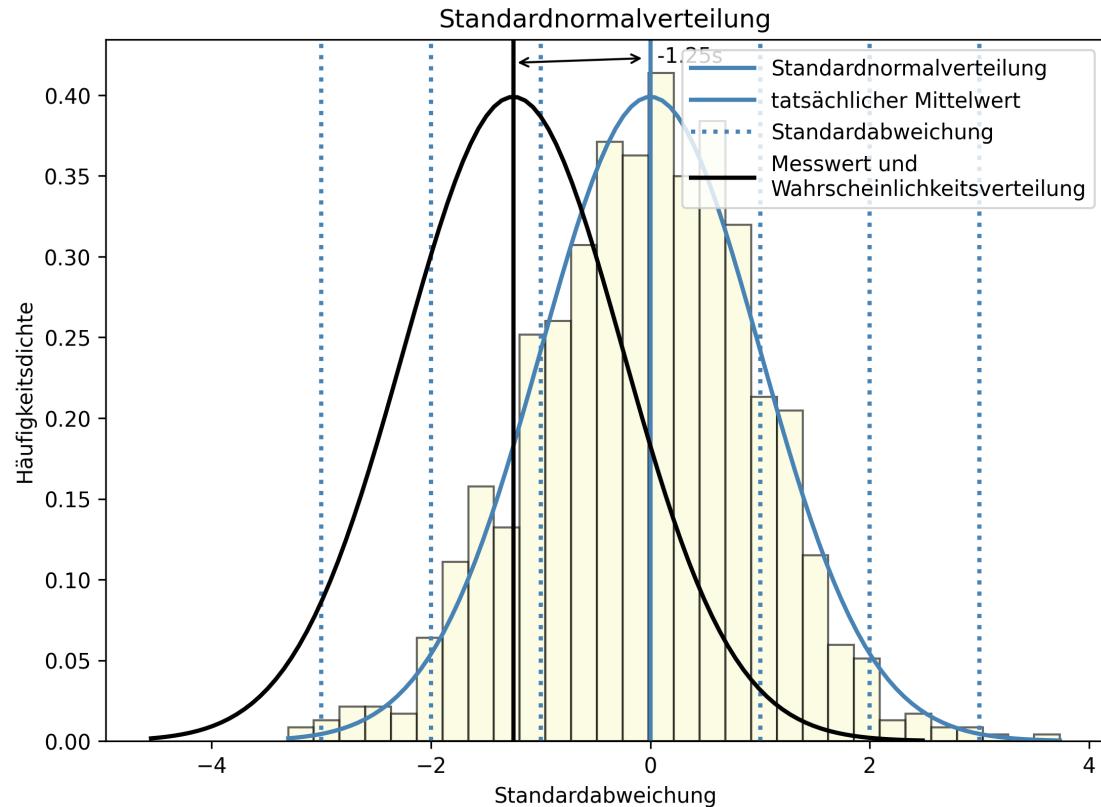
In einer Normalverteilung kommen Werte in der Nähe des Erwartungswerts häufiger vor als weit vom Erwartungswert entfernt liegende Werte. Wie häufig oder selten ein Wert relativ zum Mittelwert der Verteilung vorkommt, kann mit der Standardabweichung ausgedrückt werden.

In der Grafik sehen Sie zufällig gezogene Werte und eine Normalverteilungskurve.



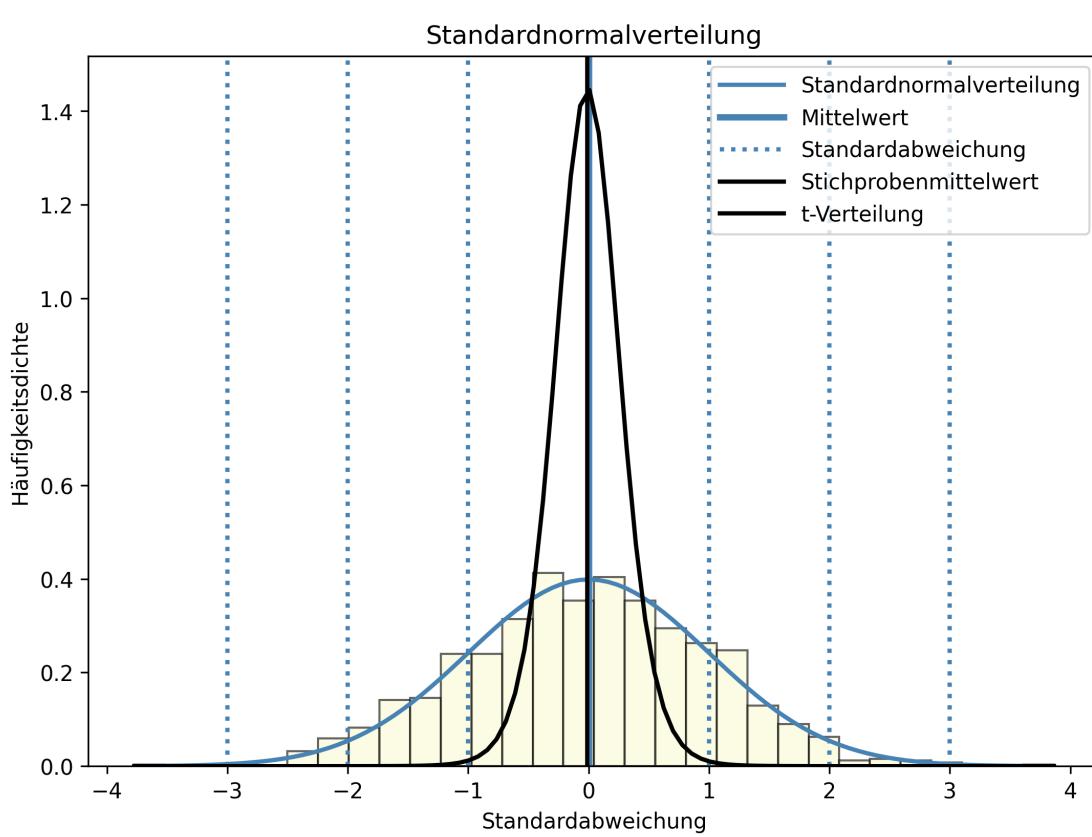
## 66.17 Einzelter Messwert

Ein einzelner Messwert aus der Verteilung entspricht so gut wie nie dem Erwartungswert. Man weiß aber, dass ein Wert nahe des Erwartungswerts häufiger vorkommt, als ein weit entfernter.



## 66.18 Stichprobe $N = 12$

Der Mittelwert einer Stichprobe entspricht ebenfalls so gut wie nie dem Erwartungswert. Der zu erwartende, in Einheiten des Standardfehlers des Mittelwerts ausgedrückte Streubereich ist jedoch erheblich schmäler als der eines einzelnen Messwerts. Dies ist auch grafisch gut zu sehen. Die Dichtekurve ist erheblich schmäler und höher als die Normalverteilungskurve eines einzelnen Messwerts. Dies liegt an der geringen Standardabweichung in der Stichprobe von  $\sim 0.2$ , was die Kurve staucht.



## 66.19 Code

Code für das Panel Stichprobe  $N = 12$

```

import numpy as np
import matplotlib.pyplot as plt
import scipy

# Parameter der Standardnormalverteilung
mu, sigma = 0, 1 # Mittelwert und Standardabweichung

# Daten generieren
seed = 4
np.random.seed(seed = seed)
data = np.random.default_rng().normal(mu, sigma, 1000)

# Grafik
plt.figure(figsize = (8.5, 6))

# Histogramm plotten
array, bins, patches = plt.hist(data, bins = 30, density = True, alpha = 0.6, color = 'lightblue')

# Mittelwert einzeichnen
mean_line = plt.axvline(mu, color = 'steelblue', linestyle = 'solid', linewidth = 3)

# positive und negative Standardabweichungen einzeichnen
pos_std_lines = [plt.axvline(mu + i * sigma, color = 'steelblue', linestyle = 'dotted', linewidth = 2) for i in range(-3, 4)]
neg_std_lines = [plt.axvline(mu - i * sigma, color = 'steelblue', linestyle = 'dotted', linewidth = 2) for i in range(3, -4, -1)]

# Normalverteilungskurve
x_values = np.linspace(min(bins), max(bins), 100)
y_values = 1 / (sigma * np.sqrt(2 * np.pi)) * np.exp(-(x_values - mu) ** 2 / (2 * sigma ** 2))
normal_dist_curve = plt.plot(x_values, y_values, color = 'steelblue', linestyle = 'solid', linewidth = 2)

# Stichprobe
N = 12
np.random.seed(seed = 4)
stichprobe = np.random.default_rng().normal(mu, sigma, N)

stichprobenstandardabweichung = stichprobe.std(ddof = 1)
stichprobenmittelwert = stichprobe.mean()
standardfehler = stichprobenstandardabweichung / np.sqrt(len(stichprobe))

# Histogramm berechnen
# hist, bins = np.histogram(stichprobe, bins = 30, density = True)

# Standardfehlerkurve Stichprobe
# x_values = np.linspace(min(bins), max(bins), 100)
x = np.linspace(stichprobenmittelwert - 4 * stichprobenstandardabweichung, stichprobenmittelwert + 4 * stichprobenstandardabweichung, 687)
y_values = scipy.stats.t.pdf(x = x, df = N - 1, loc = stichprobenmittelwert, scale = 1)

# Stichprobenmittelwert einzeichnen
mean_stichprobe = plt.axvline(stichprobenmittelwert, color = 'black', linestyle = 'solid', linewidth = 3)

# Verteilungskurve einzeichnen
stichprobe_dist_curve = plt.plot(x_values, y_values, color = 'black', linestyle = 'solid', linewidth = 2)

```

## 66.20 Die t-Verteilung

[Wikipedia](#)

Anzahl der Freiheitsgrade

! Important ??: Anzahl Freiheitsgrade

“Die Anzahl unabhängiger Information, die in die Schätzung eines Parameters einfließen, wird als Anzahl der Freiheitsgrade bezeichnet. Im Allgemeinen sind die Freiheitsgrade einer Schätzung eines Parameters gleich der Anzahl unabhängiger Einzelinformationen, die in die Schätzung einfließen, abzüglich der Anzahl der zu schätzenden Parameter, die als Zwischenschritte bei der Schätzung des Parameters selbst verwendet werden. Beispielsweise fließen  $n$  Werte in die Berechnung der Stichprobenvarianz ein. Dennoch lautet die Anzahl der Freiheitsgrade  $n - 1$ , da als Zwischenschritt der Mittelwert geschätzt wird und somit ein Freiheitsgrad verloren geht.”

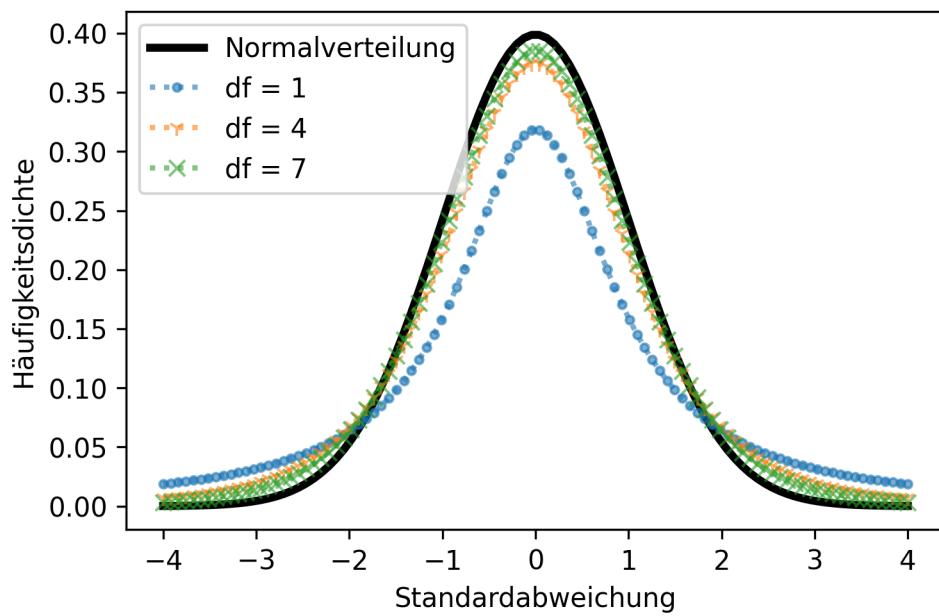
Anzahl der Freiheitsgrade (Statistik). von verschiedenen [Autor:innen](#) steht unter der Lizenz [CC BY-SA 4.0](#) ist abrufbar auf [Wikipedia](#). 2025

- 
- [Gammafunktion](#)

- 
- 
- 
- 
- 

## 66.21 Grafik

Das Argument df der t-Verteilung



## 66.22 Code

```
x_values = np.linspace(-4, 4, 100)

# Normalverteilung
y_values = scipy.stats.norm.pdf(x_values)
plt.plot(x_values, y_values, color = 'black', lw = 3, label = 'Normalverteilung')
# plt.ylim(bottom = 0, top = 0.5)

# t-Verteilungen
marker = [".", "1", "x"]

[plt.plot(x_values, scipy.stats.t.pdf(x_values, df = (i + (i - 1) * 2)), linestyle = 'dotted'

plt.suptitle('Das Argument df der t-Verteilung')
plt.xlabel('Standardabweichung')
plt.ylabel('Häufigkeitsdichte')
plt.legend(loc = 'upper left')
plt.show()
```

- 
- 
- 
- 
- 

### 💡 Tip ??: t-Verteilung oder Normalverteilung?

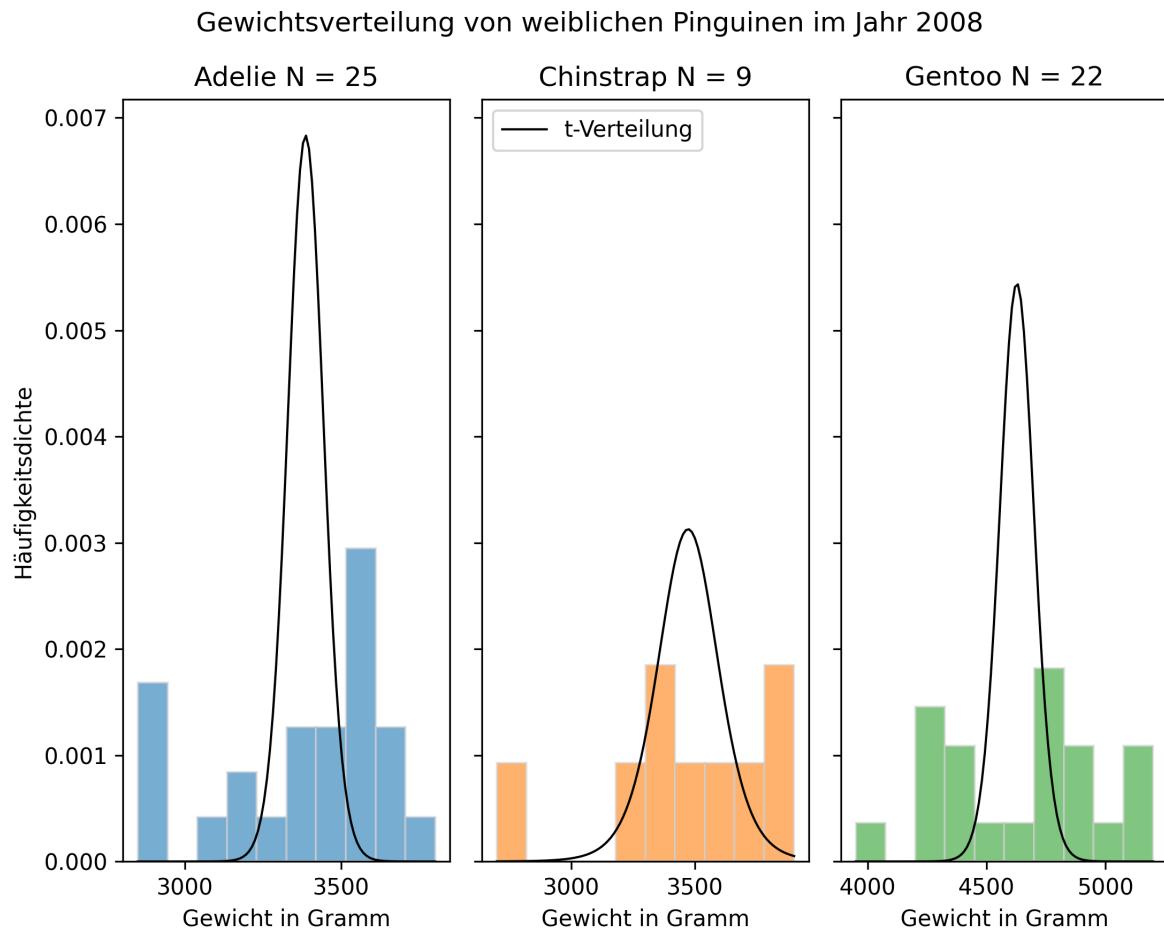
Am Computer ist die t-Verteilung genauso leicht (oder schwer) zu berechnen wie die Normalverteilung. Da die t-Verteilung bessere Schätzwerte für kleine Stichproben liefert und sich für größere Stichprobengrößen ohnehin der Normalverteilung annähert, empfiehlt es

sich, stets die t-Verteilung zu verwenden.

### 66.22.1 Beispiel Gewicht weiblicher Pinguine

```
print(penguins.groupby(by = [penguins['species'], penguins['sex'], penguins['year']]).size())
```

## 66.23 Grafik



## 66.24 Code

```
year = 2008
sex = 'female'
species = 'Adelie'

fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize = (7.5, 6), sharey = True, layout = 'tight')
plt.suptitle('Gewichtsverteilung von weiblichen Pinguinen im Jahr 2008')

# Adelie
data = penguins['body_mass_g'][ (penguins['species'] == species) & (penguins['sex'] == sex) &
```

```

stichprobengröße = data.size

## Histogramm
ax1.hist(data, alpha = 0.6, edgecolor = 'lightgrey', color = 'C0', density = True)
ax1.set_xlabel('Gewicht in Gramm')
ax1.set_ylabel('Häufigkeitsdichte')
ax1.set_title(label = str(species) + " N = " + str(stichprobengröße))

## t-Verteilung des Stichprobenmittelwerts
stichprobenmittelwert = data.mean()
stichprobenstandardabweichung = data.std(ddof = 1)
standardfehler = stichprobenstandardabweichung / np.sqrt(stichprobengröße)
hist, bin_edges = np.histogram(data)
x_values = np.linspace(min(bin_edges), max(bin_edges), 100)
y_values = scipy.stats.t.pdf(x_values, loc = stichprobenmittelwert, scale = standardfehler,)

ax1.plot(x_values, y_values, color = 'black', linewidth = 1, label = 't-Verteilung')
ax1.legend(loc = 'upper left')

# Chinstrap
species = 'Chinstrap'

data = penguins['body_mass_g'][ (penguins['species'] == species) & (penguins['sex'] == sex) &
stichprobengröße = data.size

## Histogramm
ax2.hist(data, alpha = 0.6, edgecolor = 'lightgrey', color = 'C1', density = True)
ax2.set_xlabel('Gewicht in Gramm')
ax2.set_title(label = str(species) + " N = " + str(stichprobengröße))

## t-Verteilung des Stichprobenmittelwerts
stichprobenmittelwert = data.mean()
stichprobenstandardabweichung = data.std(ddof = 1)
standardfehler = stichprobenstandardabweichung / np.sqrt(stichprobengröße)
hist, bin_edges = np.histogram(data)
x_values = np.linspace(min(bin_edges), max(bin_edges), 100)
y_values = scipy.stats.t.pdf(x_values, loc = stichprobenmittelwert, scale = standardfehler,)

ax2.plot(x_values, y_values, color = 'black', linewidth = 1)

# Gentoo
species = 'Gentoo'

```

```

data = penguins['body_mass_g'][ (penguins['species'] == species) & (penguins['sex'] == sex) &
stichprobengröße = data.size

## Histogramm
ax3.hist(data, alpha = 0.6, edgecolor = 'lightgrey', color = 'C2', density = True)
ax3.set_xlabel('Gewicht in Gramm')
ax3.set_title(label = str(species) + " N = " + str(stichprobengröße))

## t-Verteilung des Stichprobenmittelwerts
stichprobenmittelwert = data.mean()
stichprobenstandardabweichung = data.std(ddof = 1)
standardfehler = stichprobenstandardabweichung / np.sqrt(stichprobengröße)
hist, bin_edges = np.histogram(data)
x_values = np.linspace(min(bin_edges), max(bin_edges), 100)
y_values = scipy.stats.t.pdf(x_values, loc = stichprobenmittelwert, scale = standardfehler,)

ax3.plot(x_values, y_values, color = 'black', linewidth = 1)

plt.show()

```

## 66.25 Aufgabe Konfidenzintervalle

1.

2.

 Tip ??: Tipp und Musterlösung

Folgende Schritte helfen Ihnen bei der Lösung:

1. Bestimmen Sie den Stichprobenmittelwert  $\bar{x}$ .
2. Bestimmen Sie die Stichprobenstandardabweichung  $s$ , die Stichprobengröße  $N$  und den Standardfehler  $\frac{s}{\sqrt{N}}$ .
3. Bestimmen Sie die z- oder t-Werte der Normal- bzw. t-Verteilung für das gewählte Konfidenzniveau - für einen zweiseitigen Hypothesentest  $\frac{\alpha}{2}$  und  $1 - \frac{\alpha}{2}$
4. Berechnen Sie das Konfidenzintervall  $\bar{x} \pm t_{\alpha/2} \frac{s}{\sqrt{n}}$ .

### Musterlösung

Alphaniveau definieren und Pinguine auswählen

```
alpha = 1 - 0.9
data = penguins[(penguins['sex'] == sex) & (penguins['year'] == year)]
```

1. Stichprobenmittelwerte bestimmen.

```
penguin_means = data['body_mass_g'].groupby(by = data['species']).mean()
print(penguin_means)
```

```
species
Adelie      3386.000000
Chinstrap   3472.222222
Gentoo     4627.272727
Name: body_mass_g, dtype: float64
```

2. Stichprobenstandardabweichung, Stichprobengröße und Standardfehler bestimmen.

```
penguin_stds = data['body_mass_g'].groupby(by = data['species']).std(ddof = 1)
penguin_sizes = data['body_mass_g'].groupby(by = data['species']).size()
penguin_stderrors = penguin_stds / np.sqrt(penguin_sizes)

print("Stichprobenstandardabweichungen:\n", penguin_stds)
print("\nStichprobengrößen:\n", penguin_sizes)
print("\nStandardfehler:\n", penguin_stderrors)
```

Stichprobenstandardabweichungen:

```
species
Adelie      288.862712
Chinstrap   370.903551
Gentoo     339.722321
Name: body_mass_g, dtype: float64
```

Stichprobengrößen:

```
species
Adelie      25
Chinstrap   9
Gentoo     22
Name: body_mass_g, dtype: int64
```

Standardfehler:

```
species
Adelie      57.772542
Chinstrap   123.634517
Gentoo      72.429042
Name: body_mass_g, dtype: float64
```

### 3. t-Werte bestimmen

```
t_unten = scipy.stats.t.ppf(alpha / 2, loc = 0, scale = 1, df = penguin_sizes - 1)
print("t-Wert untere Intervallgrenze:", t_unten)

t_oben = scipy.stats.t.ppf(1 - alpha / 2, loc = 0, scale = 1, df = penguin_sizes - 1)
print("t-Wert obere Intervallgrenze:", t_oben)
```

```
t-Wert untere Intervallgrenze: [-1.71088208 -1.85954804 -1.7207429 ]
t-Wert obere Intervallgrenze: [1.71088208 1.85954804 1.7207429 ]
```

### 4. Konfidenzintervall bestimmen

```
# mit scipy.stats.t.ppf
untere_intervalle = scipy.stats.t.ppf(alpha / 2, loc = penguin_means, scale = penguin_
print("untere Intervallgrenzen:", untere_intervalle)

obere_intervalle = scipy.stats.t.ppf(1 - alpha / 2, loc = penguin_means, scale = pengu
print("obere Intervallgrenzen:", obere_intervalle)

print("\n'manuelle' Berechnung:\n")
# 'manuell'
print(penguin_means + t_unten * penguin_stderrors)
print(penguin_means + t_oben * penguin_stderrors)
```

```
untere Intervallgrenzen: [3287.15799233 3242.31789851 4502.64096694]
obere Intervallgrenzen: [3484.84200767 3702.12654593 4751.90448761]
```

'manuelle' Berechnung:

```
species
Adelie      3287.157992
Chinstrap   3242.317899
Gentoo      4502.640967
Name: body_mass_g, dtype: float64
```

```
species
Adelie      3484.842008
Chinstrap   3702.126546
Gentoo      4751.904488
Name: body_mass_g, dtype: float64
```

# 67 Lineare Parameterschätzung

## 67.1 Messreihe Hooke'sches Gesetz

Hooke'sche Gesetz

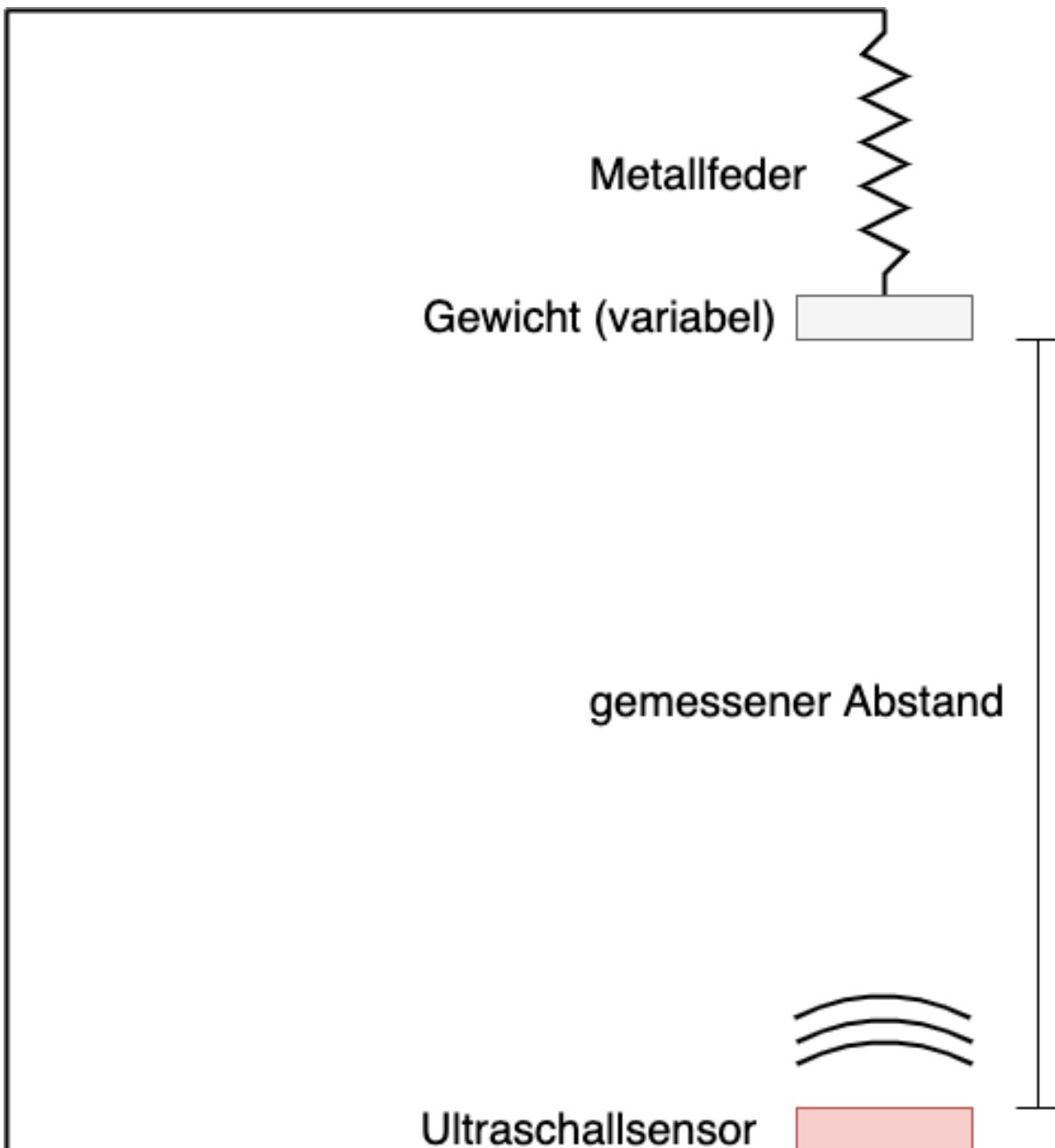


Figure 67.1: Versuchsaufbau

```
dateipfad = "01-daten/hooke_data.csv"
hooke = pd.read_csv(filepath_or_buffer = dateipfad, sep = ';')
```

### 67.1.1 Deskriptive Statistik

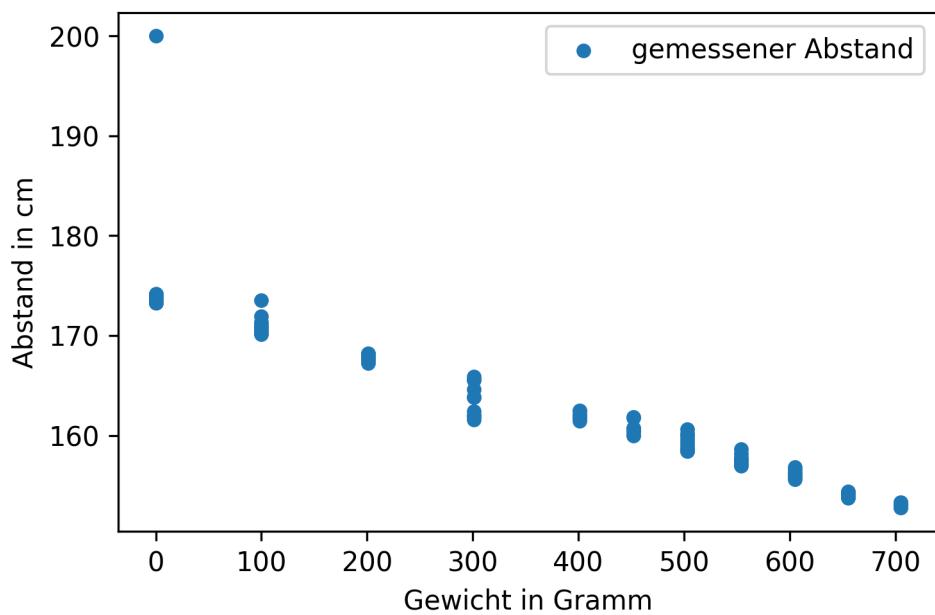
```
print(hooke.head(), "\n")
print(hooke.tail())
```

```
hooke.describe()
```

```
hooke.groupby(by = 'mass')['distance'].describe()
```

```
hooke.plot(x = 'mass', y = 'distance', kind = 'scatter', title = "Messreihe Hooke`sches Gesetz")
plt.legend()
plt.show()
```

### Messreihe Hooke`sches Gesetz



standardisiert in Einheiten der Standardabweichung (z-Werten)

```
gewicht = 0

# z-Transformation manuell berechnen
mittelwert_ausdehnung = hooke[hooke['mass'] == gewicht].loc[:, 'distance'].mean()
standardabweichung_ausdehnung = hooke[hooke['mass'] == gewicht].loc[:, 'distance'].std(ddof=1)

z_values = hooke[hooke['mass'] == gewicht].loc[:, 'distance'].apply(lambda x: (x - mittelwert_ausdehnung) / standardabweichung_ausdehnung)
z_values.name = 'z-values'

# z-Transformation mit scipy
## scipy gibt ein np.array zurück
## zur besseren Darstellung wird das np.array in eine pd.Series umgewandelt, die ein name Attribut hat
scipy_z_values = pd.Series(scipy.stats.zscore(hooke[hooke['mass'] == gewicht].loc[:, 'distance']))
scipy_z_values.name = 'scipy z-values'

# gemeinsame Ausgabe der Daten
print(pd.concat([hooke[hooke['mass'] == gewicht], z_values, scipy_z_values], axis = 1))
```

## Ausreißer

### ! Important ??: Ausreißer

In der Statistik wird ein Messwert als Ausreißer bezeichnet, wenn dieser stark von der übrigen Messreihe abweicht. In einer Messreihe können auch mehrere Ausreißer auftreten. Diese Werte können zur Verbesserung der Schätzung aus der Messreihe entfernt werden, wenn anzunehmen ist, dass diese durch Messfehler und andere Störgrößen verursacht sind.

Eine Möglichkeit, Ausreißer zu identifizieren, ist die z-Transformation. Dabei muss ein Schwellenwert gewählt werden, ab dem ein Messwert als Ausreißer klassifiziert werden soll, bspw. 2,5 oder 3 Einheiten der Standardabweichung. In der Statistik wurde eine ganze Reihe von Ausreißertests entwickelt (siehe [Ausreißertests](#))

Die Einstufung eines Messwerts als Ausreißer kann aber nicht allein auf der Grundlage statistischer Verfahren erfolgen, sondern ist immer eine Ermessensentscheidung auf der Grundlage Ihres Fachwissens. Denn nicht alle abweichenden Werte sind automatisch ungültig, sondern treten mit einer gewissen statistischen Wahrscheinlichkeit auf (siehe Kapitel Normalverteilung). Man spricht dann von gültigen Extremwerten.

Ausreißer von verschiedenen [Autor:innen](#) steht unter der Lizenz [CC BY-SA 4.0](#) und ist abrufbar auf [Wikipedia](#)

```
hooke.drop(index = 113, inplace = True)

hooke.groupby(by = 'mass')['distance'].describe()
```

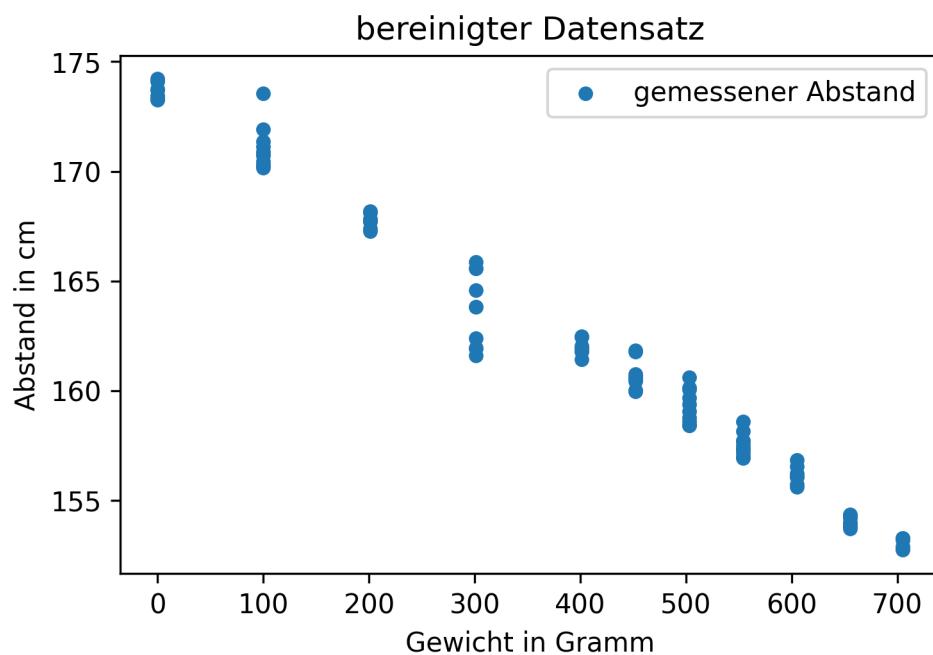
```
gewicht = 301

## scipy gibt ein np.array zurück
## zur besseren Darstellung wird das np.array in eine pd.Series umgewandelt, die ein name Attribut hat
z_values = pd.Series(scipy.stats.zscore(hooke[hooke['mass'] == gewicht].loc[:, 'distance']),
                     name = 'z-values')

print(pd.concat([hooke[hooke['mass'] == gewicht], z_values], axis = 1))
```

### 67.1.2 Explorative Statistik

```
hooke.plot(x = 'mass', y = 'distance', kind = 'scatter', title = 'bereinigter Datensatz', yla  
plt.legend()  
  
plt.show()
```

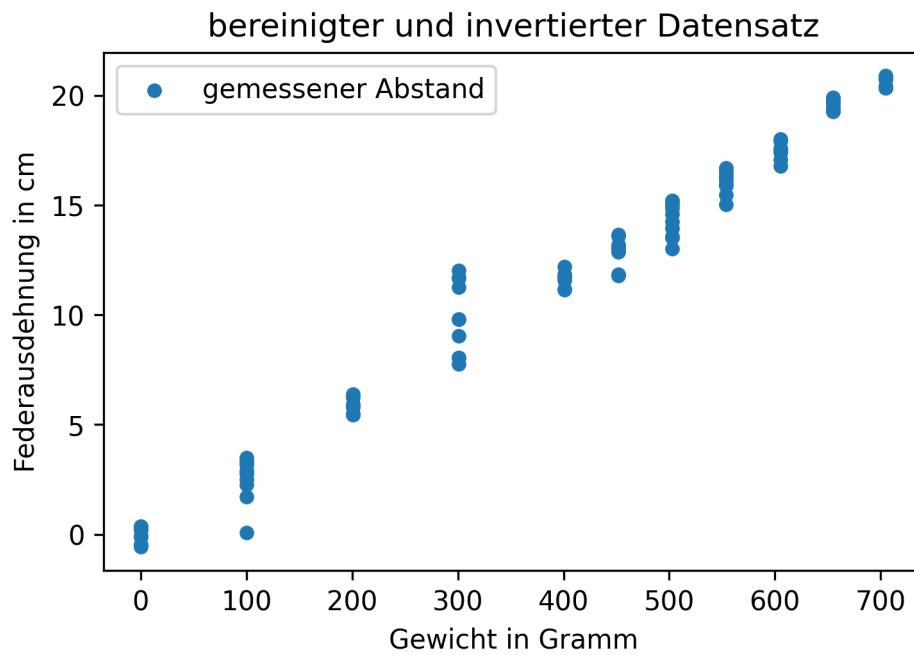


```
nullpunkt = hooke[hooke['mass'] == 0].loc[:, 'distance'].mean()  
print(f"Nullpunkt: {nullpunkt:.2f} cm")  
  
hooke['distance'] = hooke['distance'].sub(nullpunkt).mul(-1)
```

```

hooke.plot(x = 'mass', y = 'distance', kind = 'scatter', title = 'bereinigter und invertierter Datensatz')
plt.legend()
plt.show()

```



```

# Mittelwerte nach Gewicht
distance_means_by_weight = hooke['distance'].groupby(by = hooke['mass']).mean()
distance_means_by_weight.name = 'Federausdehnung'

# Standardfehler nach Gewicht
distance_stderrors_by_weight = hooke['distance'].groupby(by = hooke['mass']).std(ddof = 1)
distance_stderrors_by_weight.name = 'Standardfehler'

datenpunkte = hooke.plot(x = 'mass', y = 'distance', kind = 'scatter', title = 'bereinigter und invertierter Datensatz')

```

```

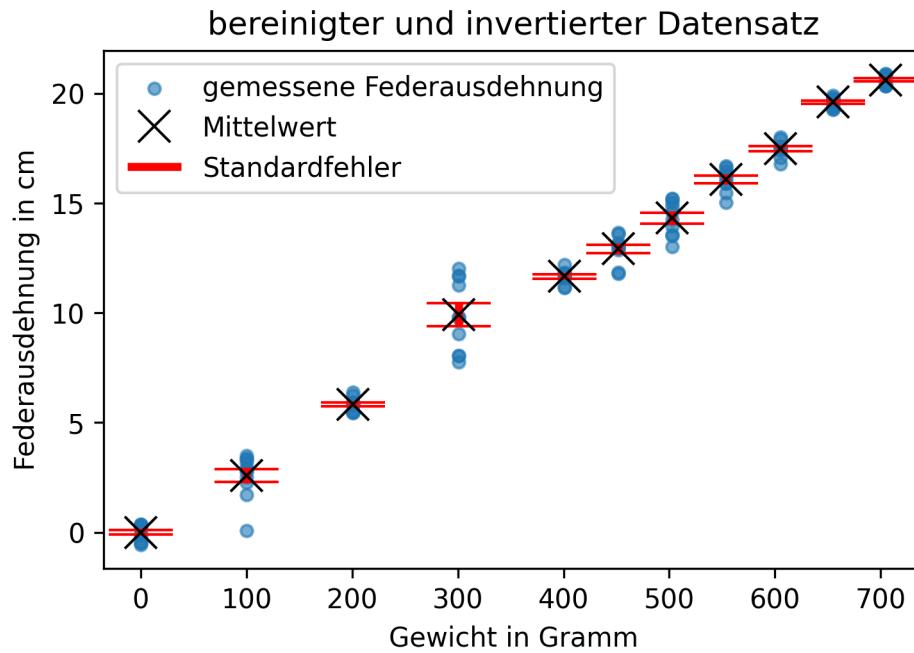
# Legendeneinträge abgreifen
# siehe: https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.get_legend_handles_labels.html
datenpunkte_handle, datenpunkte_label = datenpunkte.get_legend_handles_labels()

errorbar_container = plt.errorbar(
    x = distance_means_by_weight.index, y = distance_means_by_weight, yerr = distance_stderrors_by_weight,
    linestyle = 'none', marker = 'x', color = 'black', markersize = 12, elinewidth = 3, ecolor = 'red')

# Legende manuell erstellen
# siehe: https://matplotlib.org/stable/api/container_api.html#matplotlib.container.ErrorbarContainer
plt.legend([datenpunkte_handle[0], errorbar_container.lines[0], errorbar_container.lines[2]],[datenpunkte_label[0], 'Mittelwert', 'Standardfehler'],
           loc = 'upper left')
plt.show()

print("\n", pd.concat([distance_means_by_weight, distance_stderrors_by_weight], axis = 1))

```



## 67.2 Federkonstante bestimmen

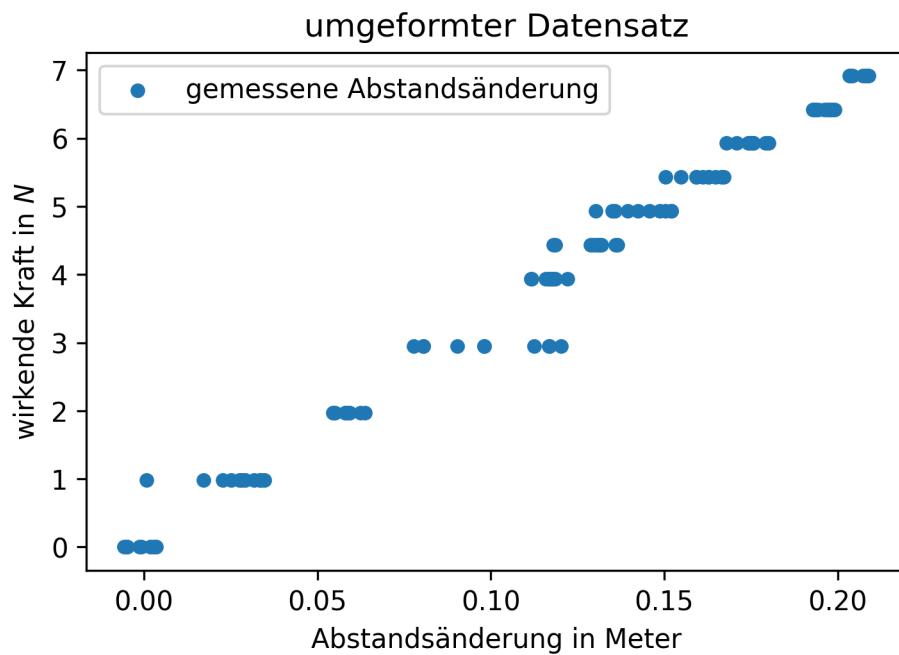
```
hooke['mass'] = hooke['mass'].div(1000).mul(9.81)
hooke.rename(columns = {'mass': 'force'}, inplace = True)

hooke['distance'] = hooke['distance'].div(100)

print(hooke.head())
```

```
hooke.plot(x = 'distance', y = 'force', kind = 'scatter', title = 'umgeformter Datensatz', yscale = 'log')
plt.legend()

plt.show()
```



### 67.2.1 Lineare Ausgleichsrechnung

Wikipedia

## Quadrate

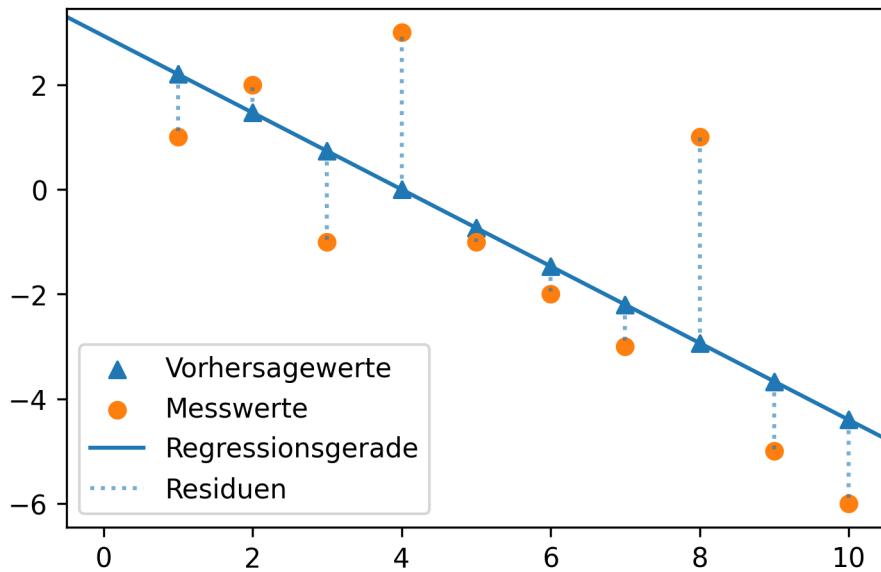
**i Note ??:** Methode der kleinsten Quadrate

Mit der Methode der kleinsten Quadrate soll diejenige Gerade  $\hat{y} = \beta_0 + \beta_1 \cdot x$  gefunden werden, die die quadrierten Abstände der Vorhersagewerte  $\hat{y}$  von den tatsächlich gemessenen Werten  $y$  minimiert. Die Werte  $y_i - \hat{y}_i$  sind die Residuen  $e_i$ . Es gilt also:

$$\sum_{i=1}^N (y_i - \hat{y}_i)^2 = \sum_{i=1}^N e_i^2 = \min$$

Grafisch kann man sich die Minimierung der quadrierten Abstände so vorstellen.

### 67.3 Grafik



Regressionskoeffizienten: [ 2.93333333 -0.73333333]

## 67.4 Code

```
x = np.arange(1, 11)
y = - x.copy() + 4
y[0] -= 2
y[2] -= 2
y[3] += 3
y[-3] += 5

lm = poly.polyfit(x, y, 1)
vorhersagewerte = poly.polyval(x, lm)

plt.scatter(x, vorhersagewerte, label = 'Vorhersagewerte', marker = "^", color = "tab:blue")
plt.scatter(x, y, label = 'Messwerte', marker = 'o', color = "tab:orange")
plt.axline(xy1 = (0, lm[0]), slope = lm[1], label = "Regressionsgerade", color = "tab:blue")
dotted = plt.vlines(x, ymin = vorhersagewerte, ymax = y, alpha = 0.6, ls = 'dotted', label = "Residuen")

plt.legend()
plt.show()

print("Regressionskoeffizienten:", lm)
```

Die eingezeichnete Gerade entspricht der linearen Funktion  $\hat{y} = \beta_0 + \beta_1 \cdot x + e_i$ . Die Dreiecksmarker sind die Vorhersagewerte  $\hat{y}_i$  des linearen Modells für die Werte  $x_i = np.arange(1, 11)$ . Die tatsächlichen Messwerte  $y$  sind mit Kreismarkern markiert. Die Länge der gestrichelten Linien entspricht der Größe der Abweichung zwischen den Mess- und Vorhersagewerten  $y_i - \hat{y}_i$ , also den Residuen  $e_i$ .

Gesucht wird diejenige Gerade, die die Summe der quadrierten Residuen minimiert. Die gesuchten Werte  $\beta_0$  und  $\beta_1$  sind die Kleinst-Quadrat-Schätzer.

$$\beta_0 = \bar{y} - \beta_1 \cdot \bar{x}$$

$$\beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x}) \cdot (y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Der Vollständigkeit halber leiten wir die Kleinst-Quadrat-Schätzer her. Gesucht werden Werte für  $\beta_0$  und  $\beta_1$ , damit die Summe der Residuenquadrate  $\sum_{i=1}^n e_i^2$  möglichst klein wird. Die Residuenquadratsumme ist die Summe der quadrierten Differenzen aus

beobachteten Werten  $y_i$  und der durch die lineare Funktion vorhergesagten Werte.

$$\sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i))^2$$

Wir untersuchen also eine Funktion, die von zwei Variablen abhängig ist.

$$f(\beta_0, \beta_1) = \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i))^2$$

Das Summenzeichen ist die Kurzschreibweise für eine Summe.

$$f(\beta_0, \beta_1) = (y_1 - (\beta_0 + \beta_1 \cdot x_1))^2 + (y_2 - (\beta_0 + \beta_1 \cdot x_2))^2 + \dots (y_n - (\beta_0 + \beta_1 \cdot x_n))^2$$

Im Minimum der Funktion müssen die beiden partiellen Ableitungen gleich Null sein  
(Warum das so ist, wird [hier](#) leicht verständlich erklärt.)

### Note 67.1: Partielle Ableitung

Die partielle Ableitung ist die Ableitung einer Funktion mit mehreren Variablen nach einer Variablen, wobei die übrigen Variablen als Konstanten behandelt werden.

Für eine Funktion  $f(x, y) = 2x + y^2$  wird die partielle Ableitung nach x so ausgedrückt:

$$\frac{\partial f(x,y)}{\partial x}$$

- Das Symbol ist die kursive Darstellung des kyrillischen Kleinbuchstaben (д) und wird als “del” gelesen. Es zeigt an, dass eine partielle Ableitung durchgeführt wird.
- Im Zähler steht die Funktion, die abgeleitet werden soll. Im Nenner steht die Variable nach der abgeleitet wird. Der Term wird gelesen als “del f von x und y nach del x”.

Die partielle Ableitung  $\frac{\partial f(x,y)}{\partial x} = 2$ .  $y^2$  wird als Konstante behandelt (z. B.  $5^2$ ) und ist abgeleitet Null.

Die partielle Ableitung  $\frac{\partial f(x,y)}{\partial y} = 2y$ .  $2x$  wird als Konstante behandelt (z. B.  $2 \cdot 3$ ) und ist abgeleitet Null.

In beiden partiellen Ableitungen sind  $x_i$  und  $y_i$  konstant. In der partiellen Ableitung nach  $\beta_0$  ist außerdem  $\beta_1$  konstant, in der partiellen Ableitung nach  $\beta_1$  ist entsprechend  $\beta_0$  konstant.

## 67.5 partielle Ableitung nach dem y-Achsenschnittpunkt

Für die partielle Ableitung nach  $\beta_0$  gilt also nach der Kettenregel für die äußere Funktion (oben) und die innere Funktion (Mitte):

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_0} = 2 \cdot (y_1 - (\beta_0 + \beta_1 \cdot x_1)) + \dots (y_n - (\beta_0 + \beta_1 \cdot x_n)) = 2 \cdot \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i)) \cdot (0 - (1 + 0 \cdot 0))$$

Für die partielle Ableitung nach  $\beta_0$  gilt also:

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_0} = -2 \cdot \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i)) = 0$$

Diese kann vereinfacht werden, indem der Vorfaktor  $-2$  entfällt (weil  $-2 \cdot 0 = 0$  gelten muss) und die Vorzeichen aufgelöst werden. Sodass:

$$\sum_{i=1}^n (y_i - \beta_0 - \beta_1 \cdot x_i) = 0$$

Man kann auch schreiben:

$$\sum_{i=1}^n y_i - \sum_{i=1}^n \beta_0 - \sum_{i=1}^n \beta_1 \cdot x_i = 0$$

$\beta_0$  und  $\beta_1$  sind Konstanten, sodass gilt  $\sum_{i=1}^n \beta_0 = \beta_0 \cdot \sum_{i=1}^n 1 = \beta_0 \cdot n$  und  $\sum_{i=1}^n \beta_1 \cdot x_i = \beta_1 \cdot \sum_{i=1}^n 1 \cdot x_i$ . So gilt:

$$\sum_{i=1}^n y_i - n \cdot \beta_0 - \beta_1 \cdot \sum_{i=1}^n x_i = 0$$

Jetzt kann man durch  $n$  teilen. Dabei entspricht  $\frac{\sum_{i=1}^n y_i}{n}$  dem arithmetischen Mittelwert von  $y$  und  $\frac{\sum_{i=1}^n x_i}{n}$  dem arithmetischen Mittelwert von  $x$ . Somit steht:

$$\bar{y} - \beta_0 - \beta_1 \cdot \bar{x} = 0$$

Umgestellt:

$$\beta_0 = \bar{y} - \beta_1 \cdot \bar{x}$$

## 67.6 partielle Ableitung nach dem Anstieg

Für die partielle Ableitung nach  $\beta_1$  ist ebenfalls die Kettenregel anzuwenden, sodass die äußere Funktion (oben) identisch abgeleitet wird:

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_1} = 2 \cdot (y_1 - (\beta_0 + \beta_1 \cdot x_1)) + \dots (y_n - (\beta_0 + \beta_1 \cdot x_n)) = 2 \cdot \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 \cdot x_i)) \cdot (0 - (0 + 1 \cdot x_i))$$

Für die partielle Ableitung nach  $\beta_1$  gilt also:

$$\frac{\partial f(\beta_0, \beta_1)}{\partial \beta_1} = -2 \sum_{i=1}^n x_i \cdot (y_i - (\beta_0 + \beta_1 \cdot x_i)) = 0$$

Auch diese kann vereinfacht werden, indem der Vorfaktor  $-2$  entfällt (weil  $-2 \cdot 0 = 0$  gelten muss) und die Vorzeichen aufgelöst werden. Außerdem kann ausmultipliziert werden:

$$\sum_{i=1}^n x_i y_i - \sum_{i=1}^n \beta_0 \cdot x_i - \sum_{i=1}^n \beta_1 \cdot x_i x_i = 0$$

Wieder können die Konstanten herausgezogen werden:

$$\sum_{i=1}^n x_i y_i - \beta_0 \cdot \sum_{i=1}^n x_i - \beta_1 \cdot \sum_{i=1}^n x_i x_i = 0$$

Jetzt kann man  $\beta_0 = \bar{y} - \beta_1 \cdot \bar{x}$  und  $\sum_{i=1}^n x_i = n \cdot \bar{x}$  einsetzen:

$$\sum_{i=1}^n x_i y_i - (\bar{y} - \beta_1 \cdot \bar{x}) \cdot n \cdot \bar{x} - \beta_1 \cdot \sum_{i=1}^n x_i x_i = 0$$

Der mittlere Term wird ausmultipliziert und  $x_i x_i$  im letzten Term als  $x_i^2$  geschrieben:

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} - \beta_1 \cdot n \bar{x} \bar{x} - \beta_1 \cdot \sum_{i=1}^n x_i^2 = 0$$

Die letzten beiden Terme werden unter Anwendung des Distributivgesetzes  $a - b = -(b - a)$  zusammengefasst.

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} - \beta_1 \cdot \left( \sum_{i=1}^n x_i^2 - n \bar{x}^2 \right) = 0$$

Jetzt kann nach  $\beta_1$  umgestellt werden. Erst:

$$\beta_1 \cdot \left( \sum_{i=1}^n x_i^2 - n \bar{x}^2 \right) = \sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}$$

Dann:

$$\beta_1 = \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{\sum_{i=1}^n x_i^2 - n \bar{x}^2}$$

Nun kann zuerst mit  $\sum_{i=1}^n x_i^2 - n \bar{x}^2 = \sum_{i=1}^n (x_i - \bar{x})^2$  umgeformt werden.

$$\beta_1 = \frac{\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Dann - und das wird gleich gezeigt - mit  $\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} = \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$ . Sodass steht:

$$\beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

Der letzte Schritt wird ausgehend vom Ergebnis gezeigt und beginnt mit dem Ausmultiplizieren:

$$\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^n (x_i y_i - x_i \bar{y} - \bar{x} y_i + \bar{x} \bar{y})$$

Man kann auch schreiben:

$$\sum_{i=1}^n x_i y_i - \sum_{i=1}^n x_i \bar{y} - \sum_{i=1}^n \bar{x} y_i + \sum_{i=1}^n \bar{x} \bar{y}$$

$\bar{x}$  und  $\bar{y}$  sind Konstanten, sodass  $\bar{x} \cdot \sum_{i=1}^n y_i$  und  $\bar{y} \cdot \sum_{i=1}^n x_i$  geschrieben werden kann.  $\sum_{i=1}^n x_i$  ist gleich  $n \cdot \bar{x}$  (analog für  $y$ ). So ergibt sich:

$$\sum_{i=1}^n x_i y_i - \bar{y} \cdot n \cdot \bar{x} - \bar{x} \cdot n \cdot \bar{y} + \sum_{i=1}^n \bar{x} \bar{y}$$

Sortieren:

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} - n \bar{x} \bar{y} + \sum_{i=1}^n \bar{x} \bar{y}$$

Der letzte Term  $\sum_{i=1}^n \bar{x} \bar{y}$  kann auch  $n \cdot \bar{x} \bar{y}$  geschrieben werden, sodass sich ergibt:

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y} - n \bar{x} \bar{y} + n \bar{x} \bar{y}$$

Die letzten beiden Terme entfallen somit und es bleibt:

$$\sum_{i=1}^n x_i y_i - n \bar{x} \bar{y}$$

[@Baitsch-2019, S. 73-74]

### 67.6.0.1 NumPy polyfit und polyeval

```
import numpy.polynomial.polynomial as poly
```

**i** Note ??: polyfit und polyeval erklärt

```
# Beispieldaten erzeugen
x = np.array(list(range(0, 100)))
y = x ** 2

print(poly.polyfit(x, y, 1))
```

```
[-1617.    99.]
```

Die Funktion gibt die geschätzten Regressionsparameter als NumPy-Array zurück. Die Terme sind aufsteigend angeordnet, d. h. der Achsabschnitt steht an Indexposition 0, der Steigungskoeffizient an Indexposition 1. Die Ausgabe für ein Polynom zweiten Grades würde beispielsweise so aussehen:

```
print(poly.polyfit(x, y, 2).round(2))
```

```
[0.  0.  1.]
```

Mit den Regressionskoeffizienten können die Vorhersagewerte der linearen Funktion berechnet werden. Dafür wird die Funktion `poly.polyval(x, c)` verwendet. Diese berechnet die Funktionswerte für in `x` übergebene Werte mit den Funktionsparametern `c`. Aus der Differenz der gemessenen Werte und der Vorhersagewerte können die Residuen bestimmt werden.

```

# 'manuelle' Berechnung
regressions_koeffizienten = poly.polyfit(x, y, 1)
vorhersagewerte = regressions_koeffizienten[0] + x * regressions_koeffizienten[1]
residuen = y - vorhersagewerte

# Berechnung mit polyeval
lm = poly.polyfit(x, y, 1)
vorhersagewerte_polyval = poly.polyval(x, lm)

print("Die Ergebnisse stimmen überein:", np.equal(vorhersagewerte, vorhersagewerte_polyval))
print("\nAusschnitt der Vorhersagewerte:", vorhersagewerte[:10])

```

Die Ergebnisse stimmen überein: True

Ausschnitt der Vorhersagewerte: [-1617. -1518. -1419. -1320. -1221. -1122. -1023. -924. -

Das Bestimmtheitsmaß  $R^2$  gibt an, wie gut die Schätzfunktion an die Daten angepasst ist. Der Wertebereich reicht von 0 bis 1. Ein Wert von 1 bedeutet eine vollständige Anpassung. Für eine einfache lineare Regression mit nur einer erklärenden Variable kann das Bestimmtheitsmaß als Quadrat des Bravais-Pearson-Korrelationskoeffizienten  $r$  berechnet werden. Dieser wird mit der Funktion `np.corrcoef(x, y)` ermittelt (die eine Matrix der Korrelationskoeffizienten ausgibt).

```

print(f"r = {np.corrcoef(x, y)[0, 1]:.2f}")
print(f"R\u00b2 = {np.corrcoef(x, y)[0, 1] ** 2:.2f}")

```

$r = 0.97$

$R^2 = 0.94$

Die Daten und die geschätzte Gerade können grafisch dargestellt werden.

```

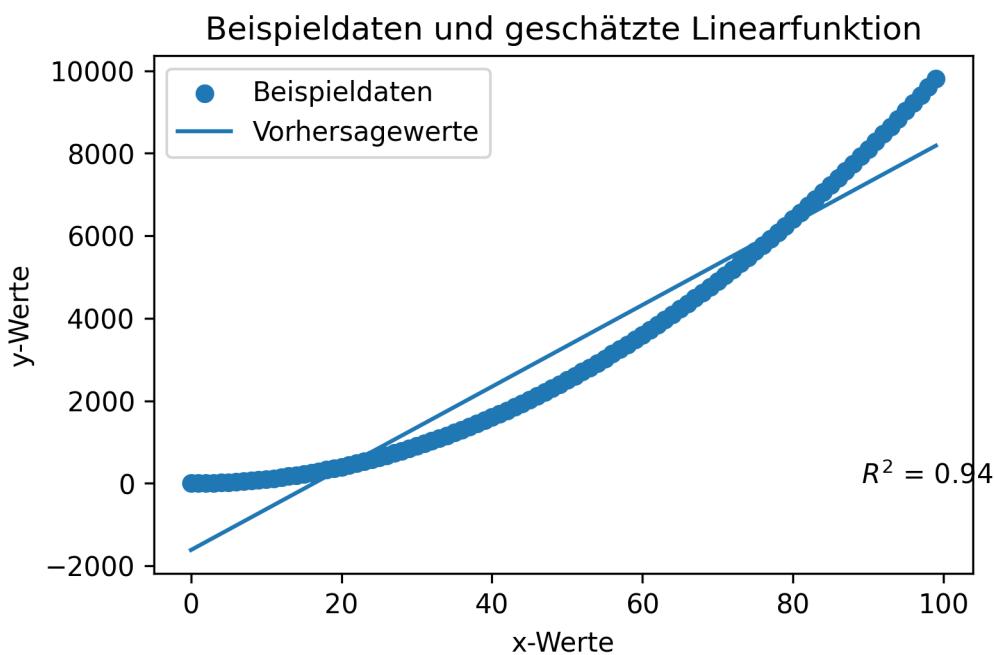
import matplotlib.pyplot as plt

plt.scatter(x, y, label = 'Beispieldaten')
plt.plot(x, vorhersagewerte, label = 'Vorhersagewerte')
plt.annotate("$R^2$ = {:.2f}".format(np.corrcoef(x, y)[0, 1] ** 2), (max(x) * 0.9, 1))

plt.title(label = 'Beispieldaten und geschätzte Linearfunktion')
plt.xlabel('x-Werte')
plt.ylabel('y-Werte')
plt.legend()

plt.show()

```



**i** Note ??: np.polyfit & np.polyval

Die Funktionen `np.polyfit(x, y, deg)` und `np.polyval(p, x)` funktionieren wie die vorgestellten Funktionen aus dem Modul `numpy.polynomial.polynomial`. Ein wichtiger Unterschied besteht jedoch darin, dass **die Parameter der Funktion polyfit in umgekehrter Reihenfolge** ausgegeben werden.

```
print(poly.polyfit(x, y, deg = 1))
print(np.polyfit(x, y, deg = 1))
```

```
[-1617.    99.]
[  99. -1617.]
```

**Note**

This forms part of the old polynomial API. Since version 1.4, the new polynomial API defined in `numpy.polynomial` is preferred. A summary of the differences can be found in the [transition guide](#).

[NumPy-Dokumentation](#)

```
print(poly.polyfit(hooke['distance'], hooke['force'], 1))

print(f'r = {np.corrcoef(hooke['distance'], hooke['force'])[0, 1]:.2f}')
print(f'R\u00b2 = {np.corrcoef(hooke['distance'], hooke['force'])[0, 1] ** 2:.2f}')
```

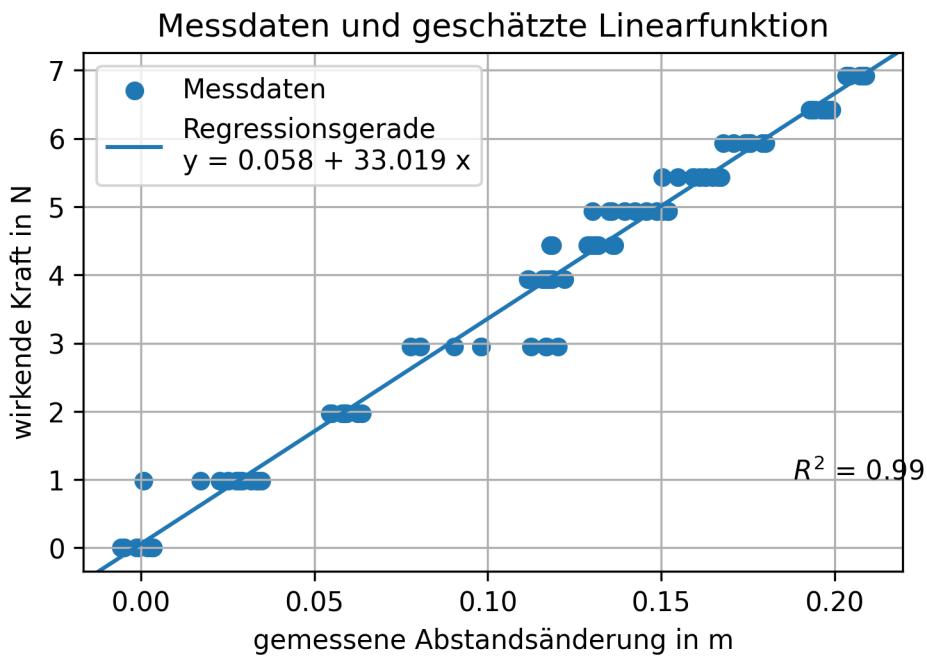
```
# Berechnung mit polyeval
lm = poly.polyfit(hooke['distance'], hooke['force'], 1)
vorhersagewerte_hooke = poly.polyval(hooke['distance'], lm)
```

```
# Platzhalter x & y
x = hooke['distance']
y = hooke['force']

# Plot erstellen
plt.scatter(x, y, label = 'Messdaten')
plt.axline(xy1 = (0, lm[0]), slope = lm[1], label = 'Regressionsgerade\ny = ' + "{beta_0:.3f} + " + "{beta_1:.3f}x")
plt.annotate("$R^2$ = {:.2f}".format(np.corrcoef(x, y)[0, 1] ** 2), (max(x) * 0.9, 1))

plt.title(label = 'Messdaten und geschätzte Linearfunktion')
plt.xlabel('gemessene Abstandsänderung in m')
plt.ylabel('wirkende Kraft in N')
plt.legend()

plt.grid()
plt.show()
```



### 67.6.1 Messabweichung quantifizieren

Wikipedia

- Residuenquadratsumme
- Summe der Abweichungsquadrate von  $x$
- $-$

```

print(f'Regressionskoeffizient: {lm[1]:.4f}')

# 'manuell' Standardfehler des Regressionskoeffizienten berechnen
standardfehler_beta_1 = np.sqrt( (1 / (len(x) - 2) * sum((y - vorhersagewerte_hooke) ** 2)) )

print(f'Standardfehler des Regressionskoeffizienten: {standardfehler_beta_1:.4f}')

# Signifikanzniveau (alpha-Niveau) 1 - 95 % wählen
alpha = 0.05
n = len(x)

t_wert = scipy.stats.t.ppf(q = 1 - alpha/2, df = n - 2)
print(f't-Wert 95-%-Intervall (zweiseitig): {t_wert:.4f}')
print(f'Konfidenzintervall 95 %: {lm[1]:.4f} ± {t_wert:.4f} * {standardfehler_beta_1:.4f}')
print(f'untere 95-%-Intervallgrenze: {lm[1] - t_wert * standardfehler_beta_1:.4f}')
print(f'obere 95-%-Intervallgrenze: {lm[1] + t_wert * standardfehler_beta_1:.4f}')

```

```

# Platzhalter x & y
x = hooke['distance']
y = hooke['force']

# Plot erstellen
plt.scatter(x, y, label = 'Messdaten')
plt.axline(xy1 = (0, lm[0]), slope = lm[1], label = 'Regressionsgerade\ny = ' + "{beta_0:.3f}" + " + " + "{beta_1:.3f}x")
plt.annotate("R^2 = {:.2f}".format(np.corrcoef(x, y)[0, 1] ** 2), (max(x) * 0.9, 1))

# 95-%-Konfidenzintervall einzeichnen
## poly.polyval(hooke['distance'], [lm[0]])
beta1_lower_boundary = lm[1] - (t_wert * standardfehler_beta_1)
beta1_upper_boundary = lm[1] + (t_wert * standardfehler_beta_1)

y_lower_boundary = poly.polyval(hooke['distance'], [lm[0], beta1_lower_boundary])

```

```

y_upper_boundary = poly.polyval(hooke['distance'], [lm[0], beta1_upper_boundary])

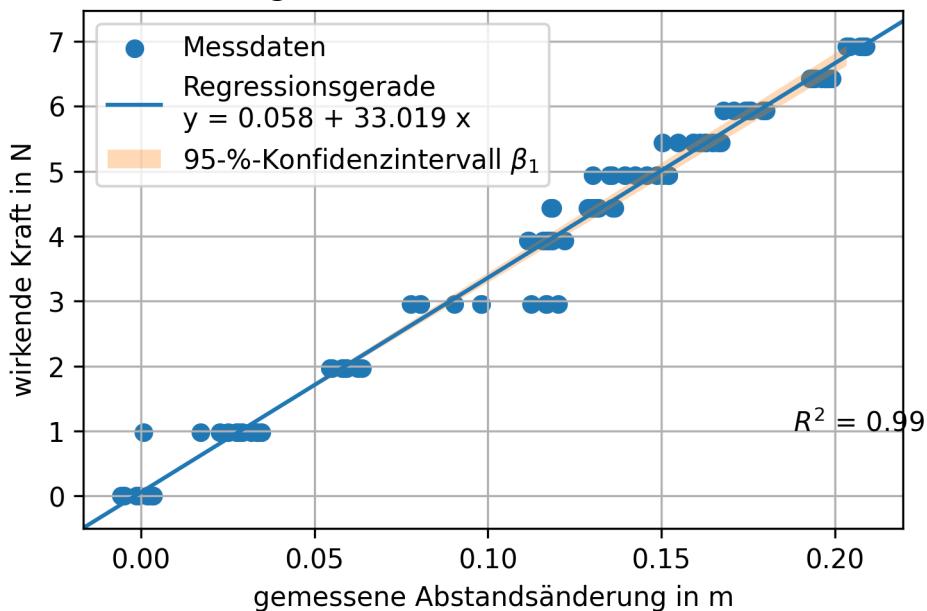
plt.fill_between(x = x, y1 = y_lower_boundary , y2 = y_upper_boundary, alpha = 0.3, label = '95-%-Konfidenzintervall')

plt.title(label = 'Messdaten und geschätzte Linearfunktion im 95-%-Intervall')
plt.xlabel('gemessene Abstandsänderung in m')
plt.ylabel('wirkende Kraft in N')
plt.legend()

plt.grid()
plt.show()

```

Messdaten und geschätzte Linearfunktion im 95-%-Intervall



### 67.6.2 Das Modul SciPy

- 
- 
- 
-

•  
•

```
# Zuweisung mehrerer Objekte
slope, intercept, rvalue, pvalue, slope_stderr = scipy.stats.linregress(x, y)
print(f"y = {intercept:.4f} + {slope:.4f} * x\n",
      f"r = {rvalue:.4f} R2 = {rvalue ** 2:.4f} p = {pvalue:.4f}\n",
      f"Standardfehler des Anstiegs: {slope_stderr:.4f}", sep = '')

# Zuweisung eines Objekts
lm = scipy.stats.linregress(x, y)

print("\n", lm, sep = '')
print(f"y-Achsen Schnittpunkt: {lm.intercept:.4f}\nStandardfehler des y-Achsen Schnittpunkts:{lm.stderr:.4f}\n")

alpha = 0.05
n = len(x)

print(f"{slope - scipy.stats.t.ppf(q = 1 - alpha / 2, df = n - 2) * slope_stderr:.3f} {slope + slope_stderr:.3f} {slope - slope_stderr:.3f}")
```

### 67.6.3 Ergebnis Federkonstante

# **68 Fehlerrechnung**

“Das Experimentieren ist ein dornenvoller Weg und so ganz sicher ist man sich nie.”  
Wagner, Paul. 2015. “Einführung in die Physik I. PH I - 04 - Die Messgenauigkeit / Fehlerrechnung Teil 1.” Universität Wien Physik, [YouTube](#), Zeitstempel 04:05

## **68.1 Bevor es losgeht**

### **68.1.1 Der Begriff des Fehlers**

- 
-

### **68.1.2 Konfidenzniveau**

“In der Physik und der Vermessungstechnik begnügt man sich mit einer statistischen Sicherheit von 68,3% und setzt deshalb  $t = 1$  [...] (In der Industrie wird  $t = 2$ , in der Biologie  $t = 3$  bevorzugt.)” [@HS-Aalen-2016 S. 4]

### **68.1.3 Notation**

## **68.2 Fehlerarten**

- - 
  - 
  - 
  -
- Quantifizierungsfehler
- - 
  -
- - 
  - 
  - 
  -
- Einschwingzeit
- - 
  -
- Parallaxenfehler
- 

💡 Tip ??: Umgang mit verschiedenen Arten von Messabweichungen

1. Grobe Fehler: Betroffene Werte streichen und Messung wiederholen.
2. Systematische Messabweichungen: Systematische Messabweichungen werden, wenn sie bekannt sind, korrigiert.
3. Zufällige Messabweichungen: Die statistische Streuung von Messwerten um den Erwartungswert wird quantifiziert.

Einige systematische und zufällige Messabweichungen können nur mit hohem Aufwand reduziert werden (genauere Messgeräte, größere Anzahl von Messvorgängen). Der Aufwand muss gegen den Gewinn an Genauigkeit abgewogen werden: Ist das Messergebnis mit der

angegebenen Unsicherheit akzeptabel?  
(@Hempel-2016, S. 2)

## 68.3 Grundbegriffe nach DIN 1319

- 
- 
- 

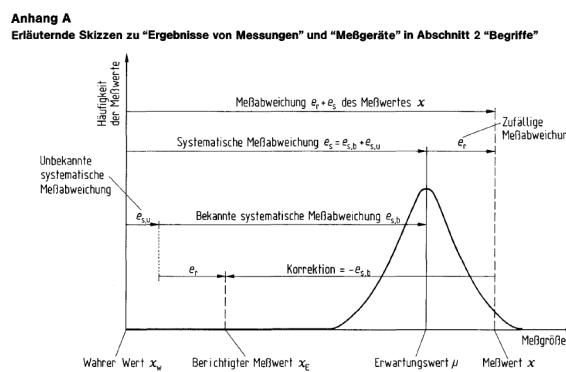


Figure 68.1: Messergebnis nach DIN 1319

- 1.
- 2.
- 3.

### 68.3.1 Messabweichung

! Important ??: Messabweichung nach DIN 1319

Die Messabweichung ist die: “Abweichung eines aus Messungen gewonnenen und der Meßgröße [...] zugeordneten Wertes vom wahren Wert [...]. Ist  $m$  der der Meßgröße zugeordnete Wert und  $x_w$  der wahre Wert, so ist die Meßabweichung des zugeordneten Wertes  $m - x_w$  [...].” [@DIN1319-1, S. 12]

- 
- 
- 

### 68.3.2 Messunsicherheit

**!** Important ??: Messunsicherheit nach DIN 1319

“Kennwert, der aus Messungen [...] gewonnen wird und zusammen mit dem Meßergebnis [...] zur Kennzeichnung eines **Wertebereiches** für den wahren Wert der Meßgröße [...] dient.” [@DIN1319-1, S. 14, eigene Hervorhebung]

Während die Messabweichung ein Einzelwert ist, kennzeichnet die Messunsicherheit einen Wertebereich: “Die Meßunsicherheit ist von der Meßabweichung [...] deutlich zu unterscheiden. Letztere ist nur die Differenz zwischen einem der Meßgröße zuzuordnenden Wert [...] und dem wahren Wert. Die Meßabweichung kann gleich Null sein, ohne daß dies bekannt ist. Diese Unkenntnis drückt sich in einer Meßunsicherheit größer als Null aus.” [@DIN1319-3, S. 3]

**i** Note ??:  $u(e_s)$  nicht vernachlässigbar

Für den Fall, dass die Unsicherheit der systematischen Messabweichung nicht als vernachlässigbar angesehen wird, verweist die DIN 1319-3 auf Berechnungsregeln in Abschnitt 6.2.5 auf die wir hier nicht eingehen. (@DIN1319-3, S. 5-6)

Die Messunsicherheit des berichtigten Messergebnisses berechnet sich aus der quadratischen Kombination der Unsicherheiten der zufälligen und der systematischen Messabweichung.

$$u(y) = \sqrt{u^2(\bar{x}) + u^2(e_s)}$$

[@DIN1319-3, Gleichung (8), S. 6, Notation angepasst]

**Warning 68.1: Notation in der DIN 1319**

In der DIN wird die zufällige Messabweichung statt mit  $u^2(\bar{x})$  mit  $u^2(x_1)$  und die Unsicherheit der systematischen Messabweichung statt mit  $u^2(e_s)$  mit  $u^2(x_2)$  notiert.

### 68.3.3 Vollständiges Messergebnis

### **⚠ Warning ??: Notation in der DIN 1319**

In der DIN wird der arithmetische Mittelwert einer Messreihe statt mit  $\bar{x}$  mit  $\bar{v}$  notiert.

#### **68.3.4 Übersicht**

##### **❗ Important ??: Grundbegriffe nach DIN 1319**

**Messabweichung:** “Abweichung eines aus Messungen gewonnenen und der Meßgröße [...] zugeordneten Wertes vom wahren Wert [...]. Ist  $m$  der der Meßgröße zugeordnete Wert und  $x_w$  der wahre Wert, so ist die Meßabweichung des zugeordneten Wertes  $m - x_w$  [...].” [@DIN1319-1, S. 12]

- Die Messabweichung des **unberichtigten Messergebnisses** setzt sich additiv aus der zufälligen Messabweichung  $e_r$  und der systematischen Messabweichung  $e_s$  zusammen.
- Die systematische Messabweichung  $e_s$  setzt sich aus einem bekannten  $e_{s,b}$  und einem unbekannten Anteil  $e_{s,u}$  zusammen.
- Die systematische Messabweichung ist die Differenz aus Erwartungswert  $\mu$  und wahrem Wert  $x_w$ .  $e_s = \mu - x_w$
- Die bekannte, systematische Messabweichung  $e_{s,b}$  wird mit entgegengesetzten Vorzeichen  $-e_{s,b}$  *Korrektion* genannt.

Das **Messergebnis**  $\bar{x}_E$  ist das um die bekannte, systematische Messabweichung  $e_{s,b}$  berichtigte arithmetische Mittel einer Messreihe  $\bar{x}$ , das als Schätzer des Erwartungswerts

$\mu$  verwendet wird.

$$\bar{x}_E = \bar{x} - e_{s,b}$$

[@DIN1319-1, S. 11-13]

**Messunsicherheit**  $u$  oder  $u(y)$ : “Kennwert, der aus Messungen [...] gewonnen wird und zusammen mit dem Meßergebnis [...] zur Kennzeichnung eines Wertebereichs für den wahren Wert der Meßgröße [...] dient.” [@DIN1319-1, S. 14]

- **Standardmessunsicherheit** oder kurz Standardunsicherheit wird verwendet, wenn die Messunsicherheit durch eine Standardabweichung ausgedrückt wird. [@DIN1319-3, S. 3]
- **erweiterte Messunsicherheit**  $U(y) = k \cdot u(y)$ : Wertebereich, der den wahren Wert der Messgröße wahrscheinlich enthält. Der Erweiterungsfaktor  $k$  sollte zwischen 2 und 3 festgelegt werden, vorzugsweise wird  $k = 2$  verwendet. Der Erweiterungsfaktor  $k$  ist mitzuteilen, damit die Standardmessunsicherheit  $u(y) = U(y)/k$  ermittelt werden kann. [@DIN1319-3, S. 7]

*Hinweis: Die erweiterte Messunsicherheit unterscheidet sich vom Konfidenzintervall dadurch, dass keine Normalverteilung unterstellt wird - es ist schlicht ein Faktor, ‘um auf Nummer sicher zu gehen’, dass der wahre Wert vom angegebenen Bereich eingeschlossen wird.*

Das **vollständige Messergebnis** ist das Messergebnis mit quantitativen Angaben zur Messunsicherheit  $u$ .  $x = \bar{x}_E \pm u$ . [@DIN1319-1, S. 16]

- Der **Vertrauensbereich**  $\bar{x} - t \cdot u(y) \leq Y \leq \bar{x} + t \cdot u(y)$  kann zusätzlich zum vollständigen Messergebnis angegeben werden.
  - Ist die systematische Messabweichung bekannt und korrigiert worden, dann liegt der wahre Wert der Messgröße  $Y$  innerhalb des Vertrauensbereichs.
  - Wenn die systematische Messabweichung nicht bekannt ist, liegt der Erwartungswert  $\mu$  der Verteilung im Vertrauensbereich. [@DIN1319-3, S. 7]

*Hinweis: In der DIN wird der arithmetische Mittelwert einer Messreihe statt mit  $\bar{x}$  mit  $\bar{v}$  geschrieben.*

## 68.4 Absolute und relative Fehler

•

•

### ⚠ Warning ??: (Alternative) Notationen

In der DIN 1319 werden Messabweichungen mit  $e$ , die Messunsicherheit mit  $u$  notiert. In der Literatur zur Fehlerrechnung wird zumeist der Begriff des (Mess-)Fehlers verwendet. Die Notation ist allerdings uneinheitlich.

- Der absolute Fehler wird häufig mit  $\epsilon_x$  (epsilon x) notiert. Üblich ist aber auch die Notation mit  $\Delta x$  (Delta), (beispielsweise [@HS-Aalen-2016; @Dinter-2011; @SchrüferEtAl2022]).  $\Delta$  wird aber ebenfalls für den Größtfehler verwendet.
- Der relative Fehler wird auch mit  $\delta x$  (delta x) notiert, bspw:  $\delta x = 4\%$ .

## 68.5 Größtfehler

### Tip ??: Größtfehler

Der Größtfehler ist keine genormte Angabe nach DIN 1319, sondern eine vereinfachte Form der Fehlerabschätzung und überschätzt die wahrscheinliche Messunsicherheit. „Für viele Betrachtungen, speziell in den Grundlagenpraktika der ersten Semester in technischen, naturwissenschaftlichen und ingenieurmäßigen Studiengängen, reicht es aus, den maximalen Fehler (*absoluter Größtfehler*) zu bestimmen.“ (@EdenGebhard-2024, S. 35)

## 68.6 Signifikante Stellen

### Important ??: signifikante Stellen

„Jede zuverlässig bekannte Stelle mit Ausnahme der Nullen, die die Position des Dezimalkommas angeben, wird signifikante Stelle genannt.“ (@EdenGebhard-2024, S. 24)  
„Gerundete numerische Unsicherheitswerte sind **mit zwei (oder bei Bedarf mit drei) signifikanten Ziffern** anzugeben. **Sie sind aufzurunden.** Das Meßergebnis ist an derselben Stelle wie die zugehörige Unsicherheit zu runden, z.B.  $y = 245,5716 \text{ mm}$  auf  $y = 245,57 \text{ mm}$ , wenn  $u(y) = 0,4528 \text{ mm}$  auf  $u(y) = 0,46 \text{ mm}$  aufgerundet wird.“ (@DIN1319-3, S. 6, eigene Hervorhebung)

1.

- 2.
- 3.
- 4.

 signifikante Stellen und wissenschaftliches Runden

Angaben zur Messunsicherheit werden zwar aufgerundet. Trotzdem an dieser Stelle: Kennen Sie den Unterschied zwischen kaufmännischem und wissenschaftlichem Runden?

<https://www.youtube.com/watch?v=s0gPyypiGOs>

Runden und signifikante Stellen (Vorkurs Mathematik) von Edmund Weitz ist abrufbar auf [YouTube](#). 2019

Python rundet wissenschaftlich:

```
print(round(2.5), round(3.5))
```

2 4

## 68.7 Methoden der Fehlerrechnung

- 1.
- 2.
- 3.

Normal

- 1.

2.

- 
- 
- 

3.

**⚠ Warning ??: Formelzeichen und Einheiten**

Im Folgenden werden identische Zeichen für Sekunde und Strecke  $s$ , Masse und Meter  $m$ , Gramm und die Konstante  $g$  sowie  $\Delta$  für die Veränderung einer Größe und den Größtfehler verwendet. Die Formeln sollten also mit großer Aufmerksamkeit gelesen werden.

#### **68.7.1 1. Fehlerabschätzung**

- 
- 
- 
-