

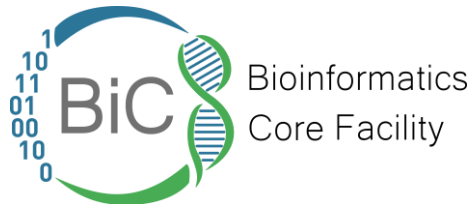
Programmieren für Einsteiger

- Arbeiten mit numerischen Daten in Python -

Tag 4

29.07.2022

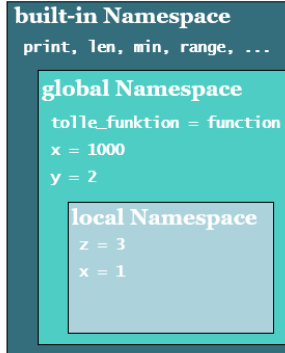
Emanuel Barth, Daria Meyer



Kurze Wiederholung Tag 3

Lokale Namensräume

- Neben dem übergeordneten *built-in Namespace* und dem *global Namespace* des Hauptprogramms, erzeugt jede Funktion ihren eigenen *local Namespace*, wenn sie aufgerufen wird.
- Namen werden immer zuerst im aktuellen Namensraum gesucht. Wird ein Name im aktuellen Namensraum nicht gefunden, dann sucht der Pythoninterpreter im nächsthöheren Namensraum.



```
def tolle_funktion(z):  
    x = 1  
    print(x + y + z)  
  
x = 1000  
y = 2  
  
tolle_funktion(3)  
6
```

13

Was ist eigentlich ein Modul?

- Ein Modul ist im Prinzip selbst nur eine normale Pythondatei, die normalen Pythoncode enthält, d. h. streng genommen ist jede beliebige Pythondatei auch ein Modul.

```
import random  
print(random)  
<module 'random' from '/home/Emanuel/miniconda3/lib/python3.7/random.py'>
```

- Man kann mit der `from import` Anweisung auch gezielt einzelne Namen zu Variablen oder Funktionen aus einem Modul importieren, ohne den Inhalt des gesamten Moduls in den *global Namespace* zu laden:

```
from random import randint  
randint(0,10)  
5
```



16

Externe Module installieren

- Normalerweise wird *pip* zusammen mit Python installiert, falls nicht: <https://pip.pypa.io/en/stable/installation/>
- *pip* ist ein Verwaltungsprogramm für externe Pythonmodule, welches diese und alle abhängigen Module automatisch herunterlädt und installiert.

```
C:\Benutzer\Emanuel> pip install PAKETNAME
```

- *pip* greift auf den PyPI (Python Package Index) zu: <https://pypi.org/>

21

Kurze Wiederholung Tag 3

Das Datei-Objekt

- Wenn wir über ein Pythonprogramm eine Datei, die auf der Festplatte unseres PC gespeichert ist, verarbeiten wollen (lesen und/oder schreiben), dann geht das immer nur über den Arbeitsspeicher.
- Beim einlesen einer Datei wird ihr Inhalt zuerst in den Arbeitsspeicher kopiert dann erhält sie vom Pythonprogramm einen Namen der in einem der Namensräume landet.
→ Es wurde ein neues Objekt erstellt, welches eine Datei (und ihren Inhalt) repräsentiert.
- Ändern wir den Inhalt einer eingelesenen Datei, so ändern wir zuerst nur den Inhalt im Arbeitsspeicher. Erst durch eine explizite Pythonanweisung werden diese Änderungen auch in die Datei auf der Festplatte übernommen.



24

Das Datei-Objekt

- Um ein Datei-Objekt zu erzeugen nutzt man die Funktion `open()`, welche zwei Argumente erwartet:

`open(Dateiname, Modus)`

String der den Pfad zur Datei beschreibt,
z. B. `"C:/Users/Emanuel/text.txt"`

String der den Bearbeitungsmodus
der Datei definiert, z. B. `"r"`

Modus	Erklärung
r	Die Datei wird ausschließlich zum Lesen geöffnet. Falls die Datei nicht existiert erhält man beim Öffnen, eine Fehlermeldung
w	Es wird eine Datei zum Schreiben erstellt. Falls eine Datei mit dem gleichen Namen schon existiert, wird deren Inhalt gelöscht und neu beschrieben.
a	Die Datei ist zum Anhängen neuer Daten bestimmt. Der bisherige Inhalt wird nicht gelöscht, es wird am Ende des Inhalts der Datei weiter geschrieben.

25

Dateien lesen und schreiben - Beispiele

- Nach dem Erzeugen eines Datei-Objekts, hängt von dessen Modus ab, welche Typfunktionen auf diesem Datei-Objekt ausgeführt werden können:

Lesemodus ('r')		Schreibmodus ('w') Erweiterungsmodus ('a')	
Typfunktion	Erklärung	Typfunktion	Erklärung
<code>read()</code>	Der gesamte Inhalt der Datei, wird als String zurückgegeben.	<code>write(x)</code>	Das Argument x wird als String in das Datei-Objekt geschrieben (aber noch nicht auf die Datei auf der Festplatte).
<code>readline()</code>	Die nächste Zeile (bis zum nächsten Zeilenumbruch) der Datei wird als String zurückgegeben.	<code>flush()</code>	Alle Änderungen am Inhalt des Datei-Objekts werden auf die Datei auf der Festplatte übertragen. Das Datei-Objekt wird <u>nicht</u> geschlossen.
<code>readlines()</code>	Der gesamte Inhalt der Datei, wird zeilenweise eingelesen und als eine Liste von Strings zurückgegeben.	<code>close()</code>	Das Datei-Objekt wird geschlossen und alle Änderungen an dessen Inhalt werden auf die Datei auf der Festplatte übertragen.
<code>close()</code>	Das Datei-Objekt wird geschlossen.		

29

Zuverlässigkeit beim Lesen und Schreiben

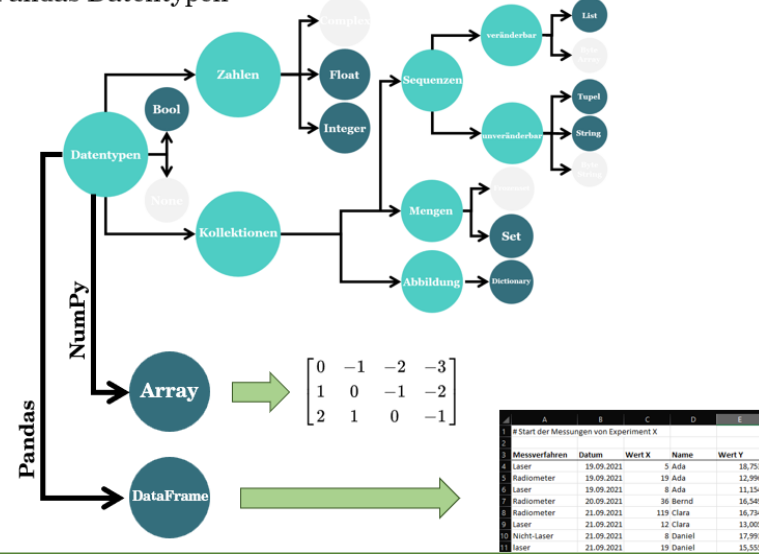
- Öffnet man ein Datei-Objekt mit der Anweisung `with` `as`, so wird das Datei-Objekt automatisch geschlossen, wenn der `with` Anweisungsblock verlassen wird:

```
with open('/home/python/wichtige_daten.dat', 'w') as output:  
    output.write('Dieses Datei-Objekt wird auch ohne close() geschlossen.')  
  
output.write('Ausserhalb des with Blocks ist das Datei-Objekt schon geschlossen')  
ValueError: I/O operation on closed file.
```

36

Kurze Wiederholung Tag 3

NumPy & Pandas Datentypen



62

Der NumPy Datentyp array

- Ein Objekt vom Datentyp array stellt eine homogene multidimensionale Matrix dar, d. h. alle Elemente in dieser Matrix haben den selben Typ (normalerweise Integer oder Float).
- Zur Erzeugung eines 1-dimensionalen array Objekts (eines Vektors) gibt es verschiedene Funktionen innerhalb des NumPy Moduls:

```
import numpy as np
x = np.array([1, 2, 3])
x
array([1, 2, 3])
type(x)
<class 'numpy.ndarray'>
```

`np.array([1, 2, 3])`

1
2
3

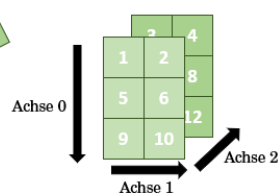
65

Mehrdimensionale Arrays

- Neben 1-dimensionalen Arrays (Vektoren), können auch relativ einfach mehrdimensionale Arrays (Matrizen) erzeugt werden. Statt von Dimensionen spricht man auch von Achsen.

```
import numpy as np
x = np.array([[1, 2],[3, 4]],[[5, 6],[7, 8]],[[9, 10],[11, 12]])
x.ndim
3
x.shape
(3, 2, 2)
np.ones((4,2,3))
```

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



`np.ones((4,2,3))`



73

Rechnen mit mehrdimensionalen NumPy Arrays

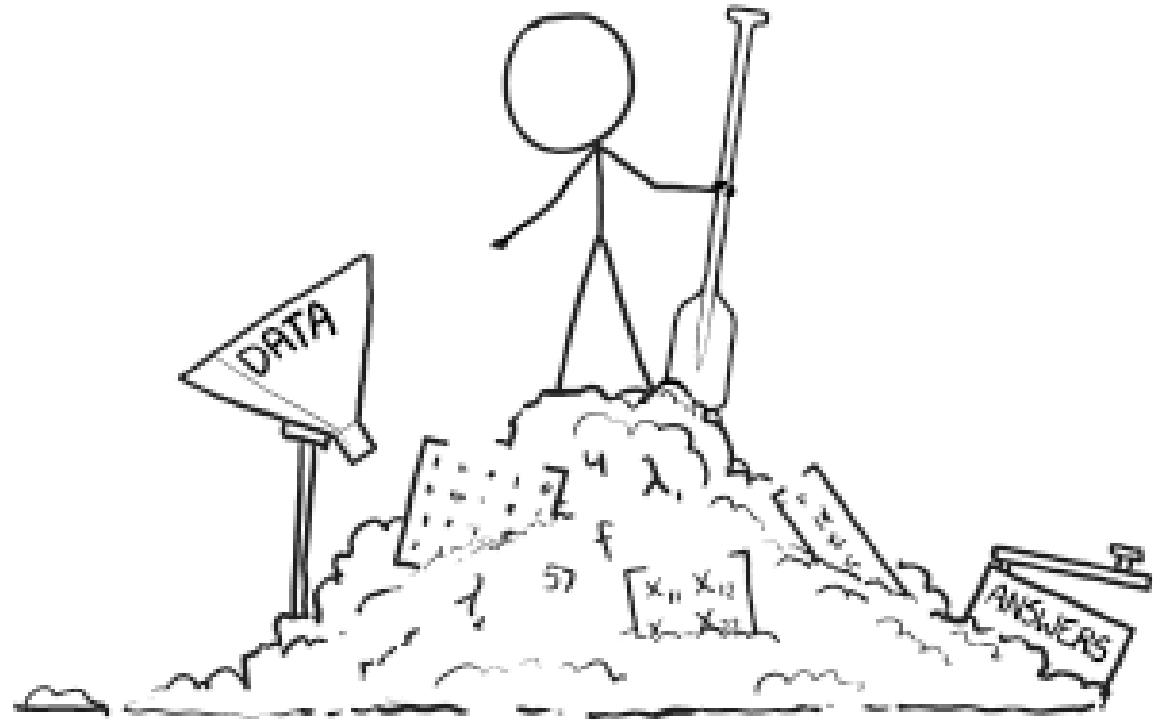
- Rechenoperationen zwischen mehrdimensionalen NumPy Arrays entsprechen den Vektor- bzw. Matrixrechenoperationen aus der Algebra:

```
import numpy as np
x = np.array([1, 2, 3, 4]).reshape(2, 2)
y = np.ones((2,2))
z = np.array([5, 6])
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$$

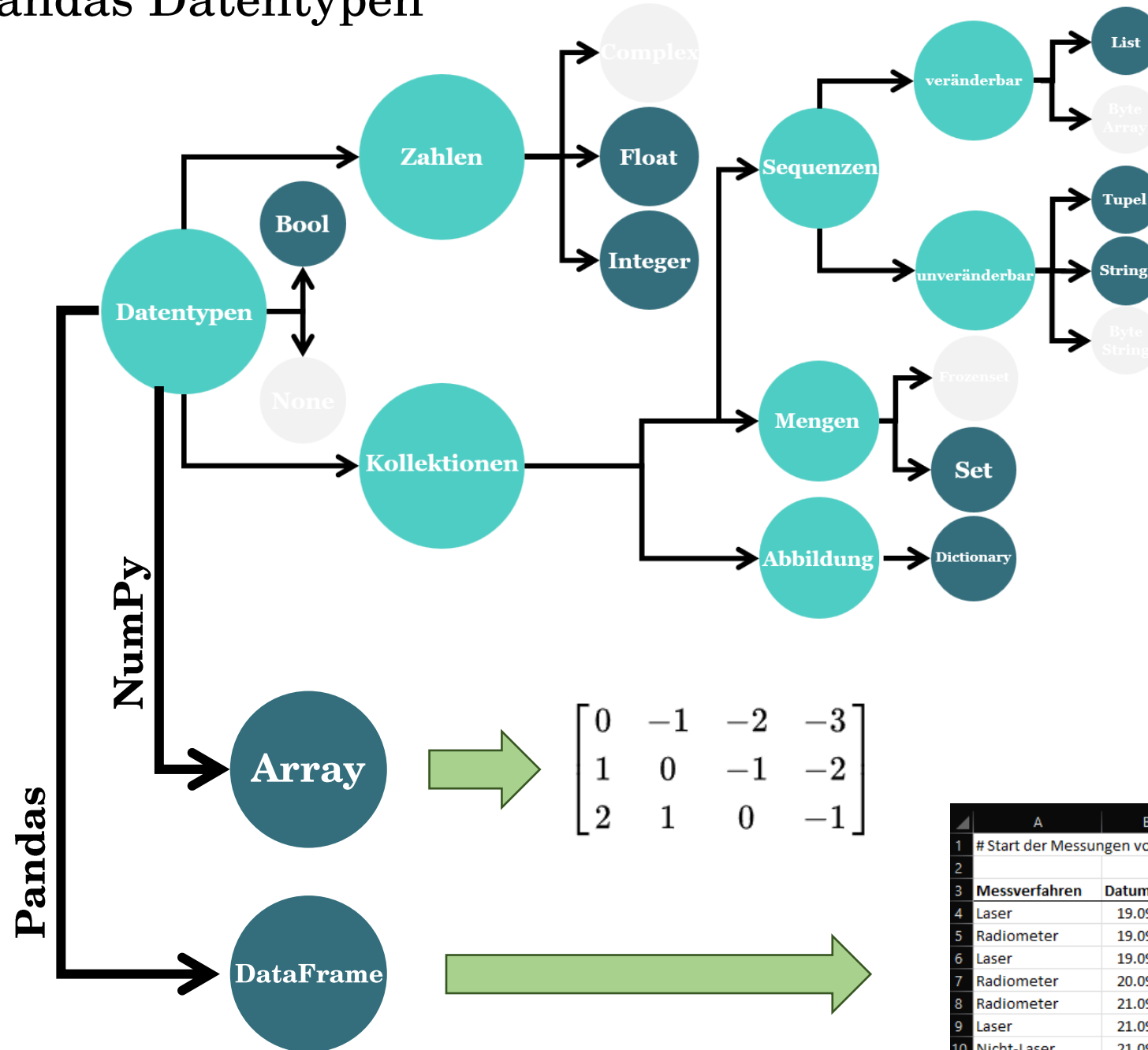
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 8 & 10 \end{bmatrix}$$

75



Pandas

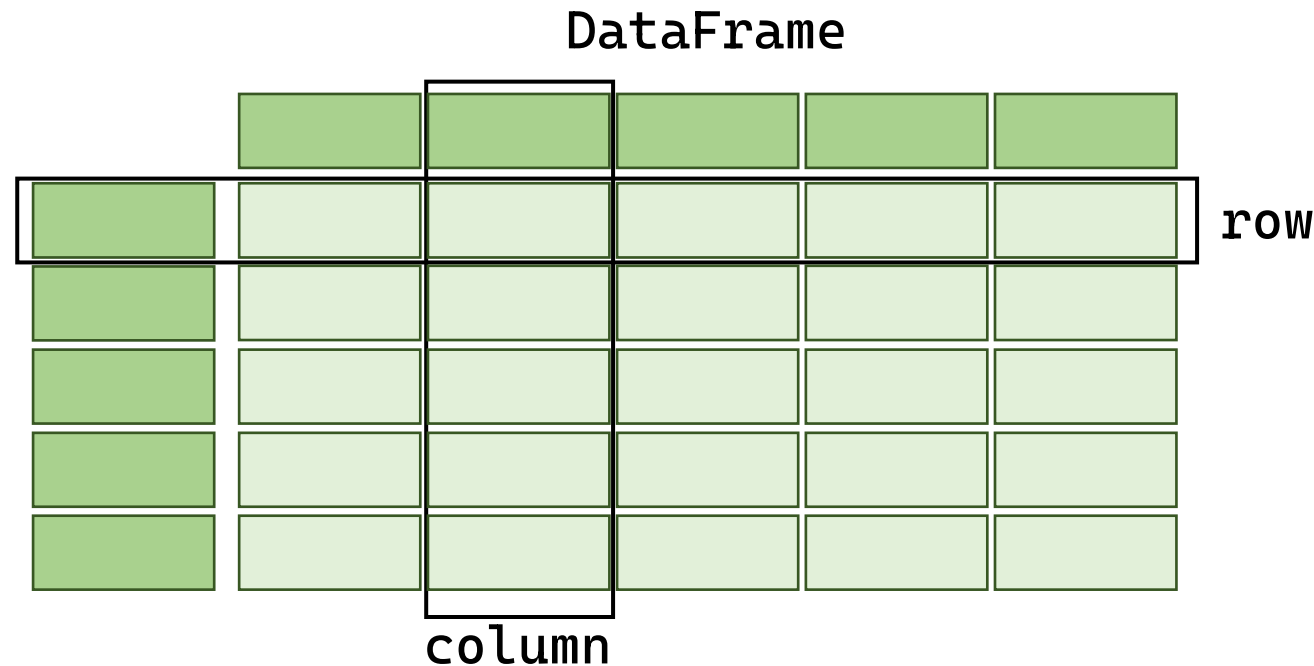
NumPy & Pandas Datentypen



	A	B	C	D	E
1	# Start der Messungen von Experiment X				
2					
3	Messverfahren	Datum	Wert X	Name	Wert Y
4	Laser	19.09.2021	5	Ada	18,753
5	Radiometer	19.09.2021	19	Ada	12,996
6	Laser	19.09.2021	8	Ada	11,154
7	Radiometer	20.09.2021	36	Bernd	16,549
8	Radiometer	21.09.2021	119	Clara	16,734
9	Laser	21.09.2021	12	Clara	13,005
10	Nicht-Laser	21.09.2021	8	Daniel	17,991
11	laser	21.09.2021	19	Daniel	15,555

Das Modul Pandas

- Pandas erweitert Python um den **DataFrame** Datentyp, welcher im Prinzip eine Tabelle mit Zeilen (rows) und Spalten (columns) darstellt, in welcher sich (fast) beliebige Daten verwalten lassen.
- Darüber hinaus besitzen **DataFrame** Objekte eine Vielzahl an Typfunktionen um die in ihnen gespeicherten Daten zu verarbeiten und zu analysieren.



DataFrame Objekte erzeugen

- Ein DataFrame Objekt lässt sich über Dictionaries erzeugen, wobei die Schlüssel als Spaltennamen und die Werte als Spalteninhalte interpretiert werden:

```
import pandas as pd
```

```
datenDictionary = {'person': ('Ada', 'Bernd', 'Carina', 'Daniel'),  
                  'geräte': ('Laser', 'Laser', 'Thermometer', 'Laser'),  
                  'wert': [5.19, 17.2, 36.0, 11.49]}
```

```
df = pd.DataFrame(datenDictionary)
```

```
print(df)
```

```
  person  geräte  wert  
0    Ada    Laser  5.19  
1   Bernd    Laser 17.20  
2  Carina Thermometer 36.00  
3  Daniel    Laser 11.49
```

	person	gerät	wert
0	Ada	Laser	5.19
1	Bernd	Laser	17.20
2	Carina	Therm.	36.00
3	Daniel	Laser	11.49

	A	B	C	D
1	person	geräte	wert	
2	Ada	Laser	5.19	
3	Bernd	Laser	17.20	
4	Carina	Thermomete	36.00	
5	Daniel	Laser	11.49	
6				
7				

Series Objekte erzeugen

- Es lassen sich auch einzelne Spalten als so genannte **Series** Objekte erzeugen:

```
import pandas as pd

x = pd.Series(['Jan', 'Jan', 'Feb', 'Apr'], name='monat')

print(x)
0    Jan
1    Jan
2    Feb
3    Apr
Name: monat, dtype: object
```

0	Jan
1	Jan
2	Feb
3	Apr

Series Objekte zu DataFrames hinzufügen

- Ein DataFrame Objekt besitzt zwar eine `append` Funktion, dieser ist aber für das Hinzufügen von Zeilen und nicht von Spalten gedacht:

```
x = pd.Series(['Jan', 'Jan', 'Feb', 'Apr'], name='monat')
```

```
df.append(x)
```

person	geräte	wert	0	1	2	3	
0	Ada	Laser	5.19	NaN	NaN	NaN	NaN
1	Bernd	Laser	17.20	NaN	NaN	NaN	NaN
2	Carina	Thermometer	36.00	NaN	NaN	NaN	NaN
3	Daniel	Laser	11.49	NaN	NaN	NaN	NaN
monat	NaN	NaN	NaN	Jan	Jan	Feb	Apr

- Stattdessen nutzt man eine ähnliche Syntax wie bei einem Dictionary:

```
df['monat'] = x
```

```
print(df)
```

	person	geräte	wert	monat
0	Ada	Laser	5.19	Jan
1	Bernd	Laser	17.20	Jan
2	Carina	Thermometer	36.00	Feb
3	Daniel	Laser	11.49	Apr

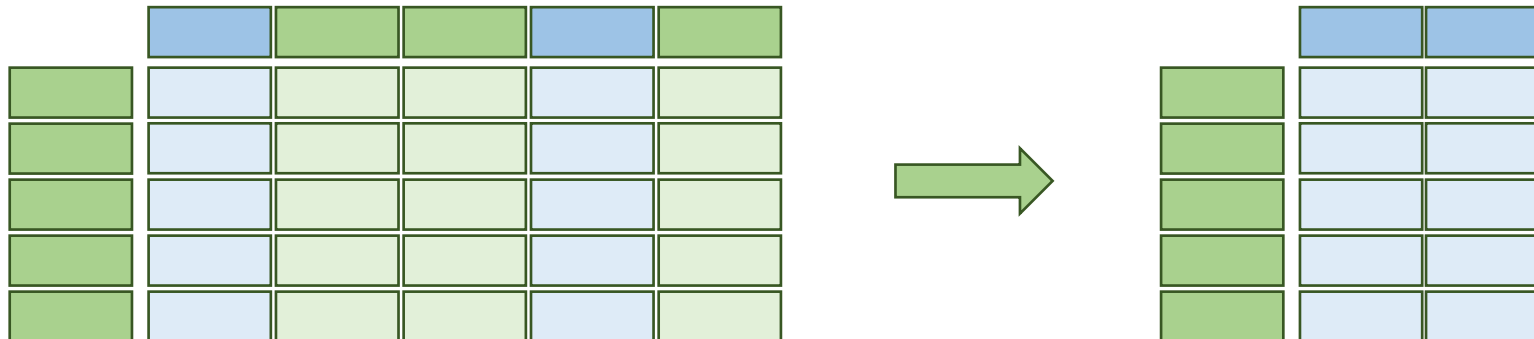
Zugriff innerhalb eines DataFrame

- Der Zugriff auf eine einzelne Spalte erzeugt ein separates **Series** Objekt mit den entsprechenden Daten:

```
df['person']  
0      Ada  
1    Bernd  
2   Carina  
3   Daniel
```

- Der Zugriff auf mehrere Spalten erzeugt ein separates **DataFrame** Objekt mit den entsprechenden Daten:

```
df[['person', 'monat']]  
  person monat  
0     Ada   Jan  
1   Bernd   Jan  
2  Carina   Feb  
3  Daniel   Apr
```



Zugriff innerhalb eines DataFrame

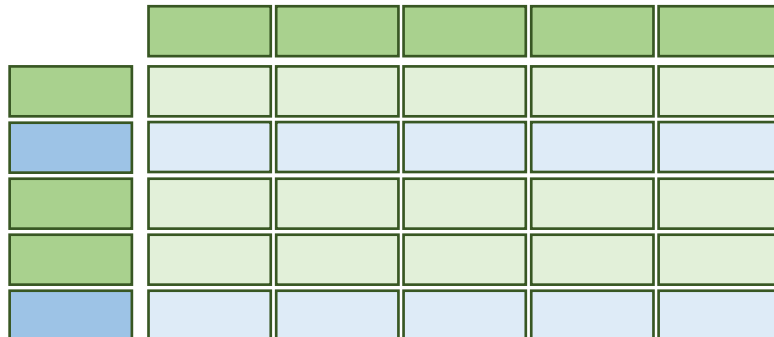
- Der Zugriff auf eine einzelne Zeilen wird meist über das Filtern nach bestimmten Datenwerten erreicht, wobei wieder ein separates DataFrame Objekt mit den entsprechenden Daten erzeugt wird:

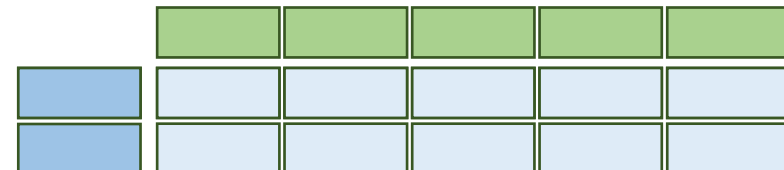
```
df[df['monat'] == 'Jan']  
person geräte wert monat  
0 Ada Laser 5.19 Jan  
1 Bernd Laser 17.20 Jan
```

- Komplexere Filter lassen sich über die `DataFrame.loc` Anweisung formulieren, wobei das `&` Zeichen für das logische *und*, das `|` Zeichen für das logische *oder* steht:

```
df.loc[(df['monat'] == 'Jan') & (df['wert'] > 6)]  
person geräte wert monat  
1 Bernd Laser 17.2 Jan
```

```
df.loc[(df['geräte'] == 'Thermometer') | (df['person'] == 'Carina')]  
person geräte wert monat  
1 Bernd Laser 17.2 Jan
```





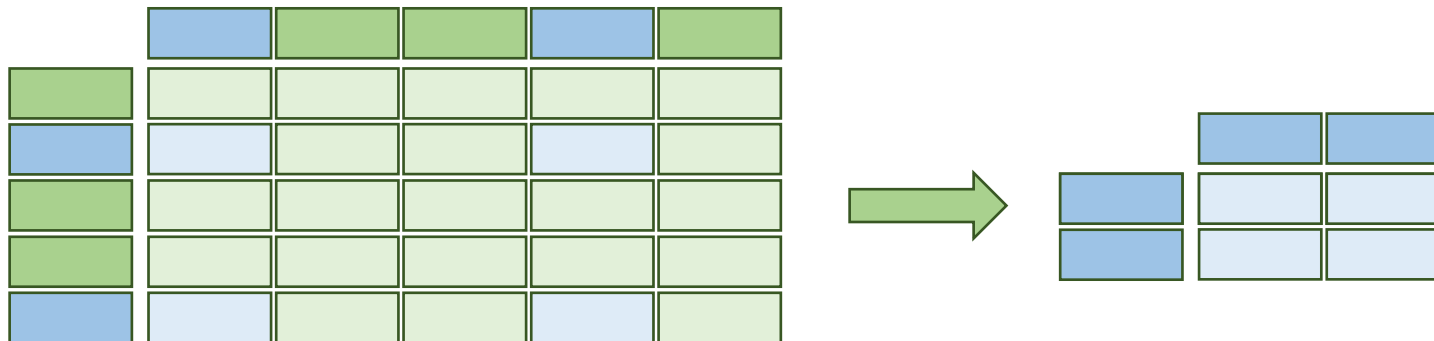
Zugriff innerhalb eines DataFrame

- Mit der `DataFrame.loc` Anweisung lässt sich auch gezielt nach Spalten und Zeilen gleichzeitig filtern:

```
df.loc[df['wert'] > 10, 'person']  
1      Bernd  
2      Carina  
3      Daniel
```

```
df.loc[(df['wert'] > 10) & (df['geräte'] == 'Laser'), ['person', 'monat']]  
   person monat  
1   Bernd   Jan  
3  Daniel   Apr
```

- Achtung: `loc` ist keine Typfunktion, deshalb immer eckige statt runde Klammern benutzen!



Zugriff innerhalb eines DataFrame

- Mit der `DataFrame.iloc` Anweisung kann man Daten auch über die Spalten- und Zeilenindizes abrufen:

```
print(df)
  person  geräte  wert  monat
0    Ada   Laser  5.19    Jan
1  Bernd   Laser 17.20    Jan
2  Carina Thermometer 36.00   Feb
3  Daniel   Laser 11.49    Apr

df.iloc[0, 1]
5.19

df.iloc[1:3, 1:3]
  geräte  wert
1   Laser  17.2
2 Thermometer 36.0
```

- Achtung: `iloc` ist keine Typfunktion, deshalb immer eckige statt runde Klammern benutzen!**

`df.iloc[0, 1]`

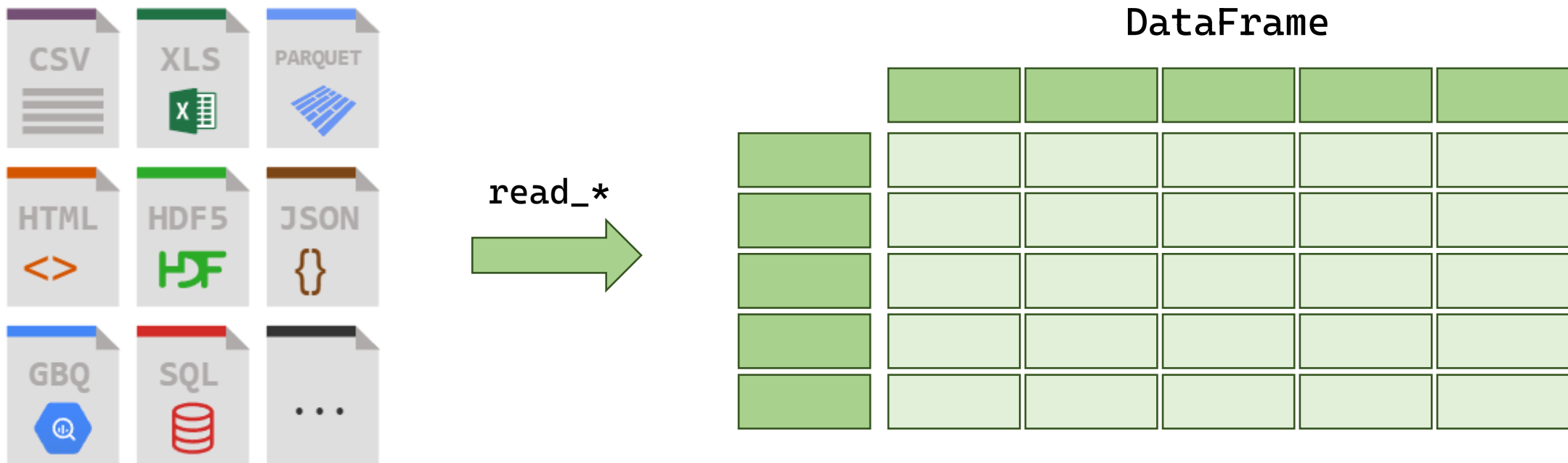
		0	1	2	3
0					
1					
2					
3					

`df.iloc[1:3, 1:3]`

		0	1	2	3
0					
1					
2					
3					

DataFrames aus vorhandenen Tabellen erzeugen

- Pandas erlaubt auch die Erstellung eines **DataFrame** Objekts aus einer großen Zahl unterschiedlicher Tabellendateiformaten wie z. B. CSV, XLS, JSON, ...
- Für jedes unterstützte Dateiformat existiert eine eigene **read** Funktion, welche die entsprechende Datei automatisch umwandelt:



DataFrames aus vorhandenen Tabellen erzeugen

- Für jedes unterstützte Dateiformat existiert eine eigene `read` Funktion, welche die entsprechende Datei automatisch umwandelt:

```
datenExp1 = pd.read_csv('./uebungsmaterial/beispieldaten/Experiment_1.csv')
```

```
print(datenExp1)
```

	Messgerät	Messwert	Person	Monat
0		Laser	10.310019	Daniel Jan
1	Thermometer	27.000000	Carina	Jan
2	Laser	18.739633	Ada	Jan
3	Laser	41.538534	Bernd	Jan
4	Laser	28.985484	Daniel	Jan
...
1063	Thermometer	86.162045	Ada	Dez
1064	Laser	5.670844	Bernd	Dez
1065	Radiometer	4.887097	Daniel	Dez
1066	Thermometer	21.252903	Ada	Dez
1067	Laser	17.604905	Daniel	Dez

```
[1068 rows x 4 columns]
```

DataFrames aus vorhandenen Tabellen erzeugen

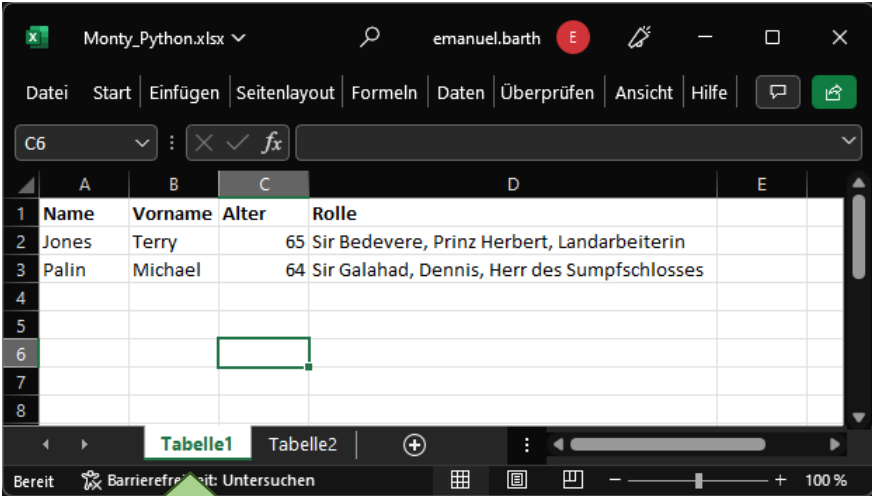
- Bei Excel-Dateien kann Pandas auch auf die einzelnen *Sheets* zugreifen:

```
pd.read_csv('./uebungsmaterial/Monty_Python.xlsx')
```

	Name	Vorname	Alter	Rolle
0	Jones	Terry	65	Sir Bedevere, Prinz Herb
1	Palin	Michael	64	Sir Galahad, Dennis, Her

```
pd.read_csv('./uebungsmaterial/Monty_Python.xlsx',  
sheet_name='Tabelle2')
```

	Name	Vorname	Alter	Rolle
0	Chapman	Graham	48	König Arthur, Wächter, S
1	Cleese	John	66	Sir Lancelot, Tim der Za
2	Gilliam	Terry	65	Knappe Patsy, Sir Bors,
3	Idle	Eric	63	Sir Robin, Diener Concor



Monty_Python.xlsx

emanuel.barth

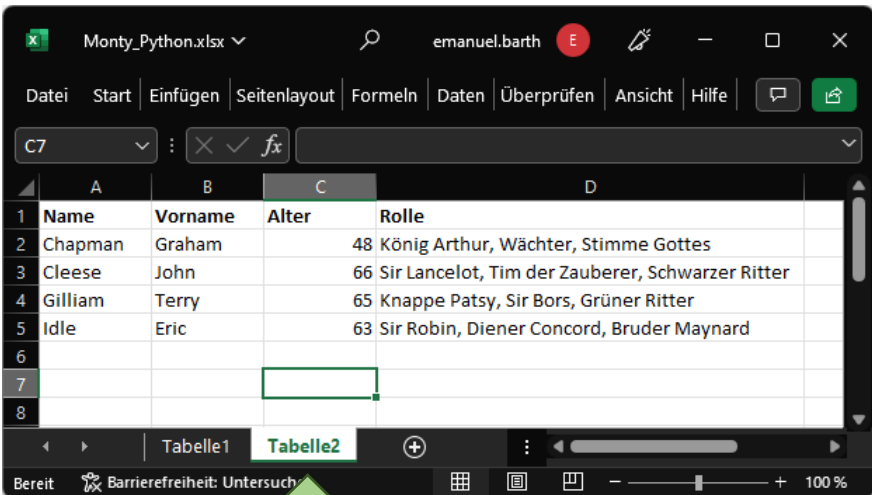
Datei Start Einfügen Seitenlayout Formeln Daten Überprüfen Ansicht Hilfe

C6

	A	B	C	D	E
1	Name	Vorname	Alter	Rolle	
2	Jones	Terry	65	Sir Bedevere, Prinz Herbert, Landarbeiterin	
3	Palin	Michael	64	Sir Galahad, Dennis, Herr des Sumpfschlosses	
4					
5					
6					
7					
8					

Tabelle1 Tabelle2

Bereit Barrierefreiheit: Untersuchen



Monty_Python.xlsx

emanuel.barth

Datei Start Einfügen Seitenlayout Formeln Daten Überprüfen Ansicht Hilfe

C7

	A	B	C	D
1	Name	Vorname	Alter	Rolle
2	Chapman	Graham	48	König Arthur, Wächter, Stimme Gottes
3	Cleese	John	66	Sir Lancelot, Tim der Zauberer, Schwarzer Ritter
4	Gilliam	Terry	65	Knappe Patsy, Sir Bors, Grüner Ritter
5	Idle	Eric	63	Sir Robin, Diener Concord, Bruder Maynard
6				
7				
8				

Tabelle1 Tabelle2

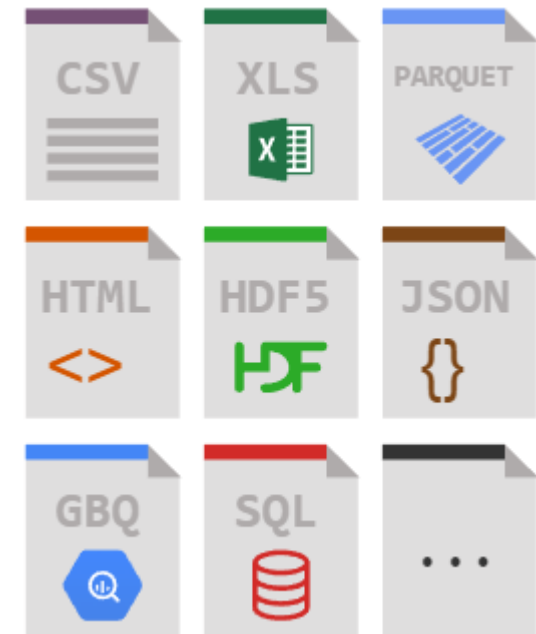
Bereit Barrierefreiheit: Untersuchen

DataFrames speichern

- Pandas kann auch **DataFrame** Objekte in Form eines Tabellendateiformaten wie z. B. CSV, XLS, JSON, ... auf der Festplatte speichern.
- Für jedes unterstützte Dateiformat existiert eine eigene **to** Typfunktion, welche ein **DataFrame** Objekt automatisch umwandelt und abspeichert:

DataFrame

to_*



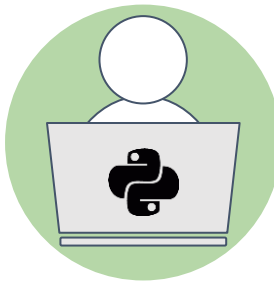
DataFrames speichern

- Für jedes unterstützte Dateiformat existiert eine eigene `to` Typfunktion, welche ein DataFrame Objekt automatisch umwandelt und abspeichert:

```
datenExp1.to_excel('./uebungsmaterial/beispieldaten/Experiment_1.xlsx')
```

- Solltes es beim Lesen oder Speichern von Excel-Dateien mit Pandas Probleme geben, muss meistens nur das *openpyxl* Modul nachinstalliert werden:

```
C:\Benutzer\Emanuel> pip install openpyxl
```



Aufgabe:

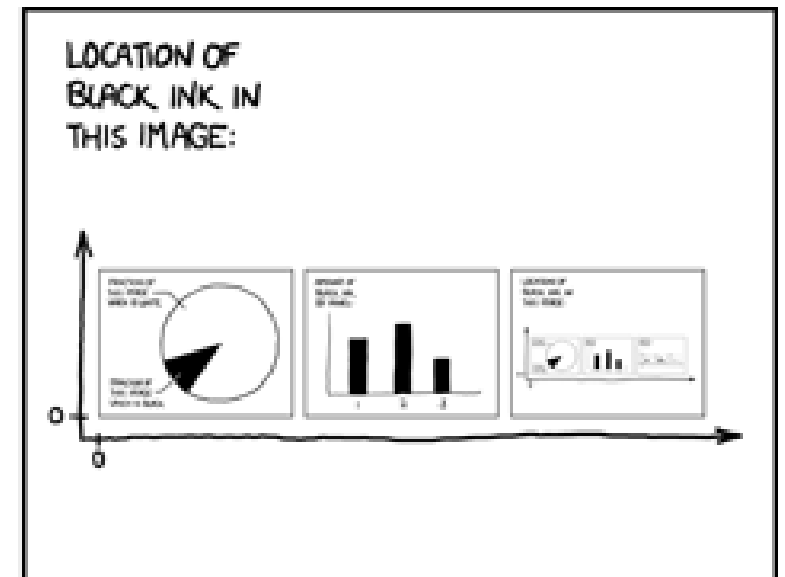
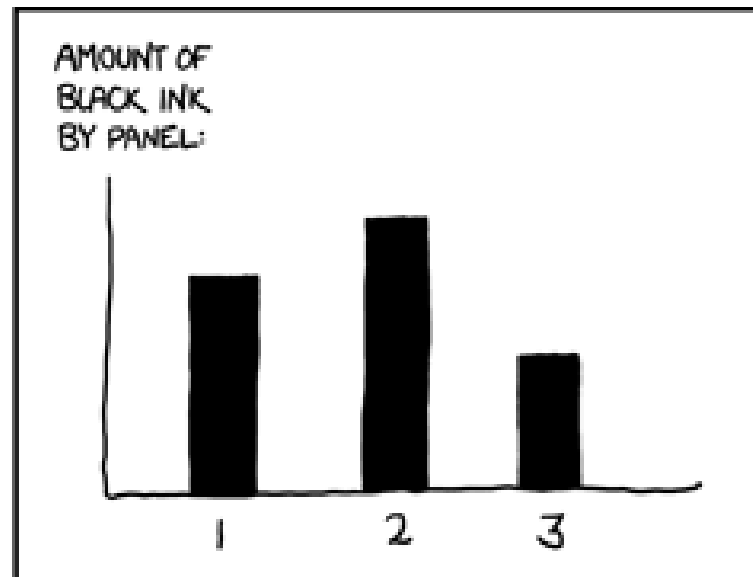
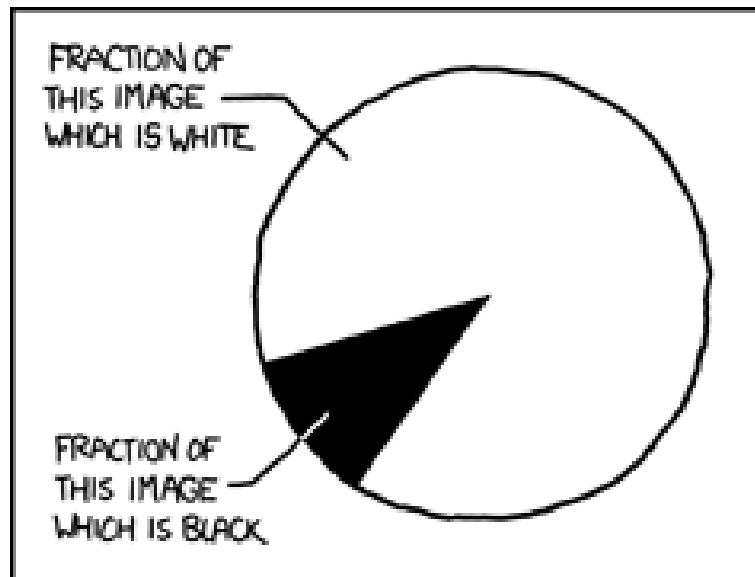
Die `DataFrame` Objekte von Pandas besitzen natürlich auch jede Menge verschiedene Typfunktionen, um vor allem numerische Werte zu verarbeiten.

Erzeugt aus einem der "Experiment_X.csv" Dateien aus eurem "beispieldaten" Ordner ein Pandas `DataFrame` und probiert die Typfunktionen `min`, `max` und `mean` aus:

```
df['wert'].min()
```

Schreibt ein Programm, welches folgende Fragen über die Experiment-Datensätze beantwortet:

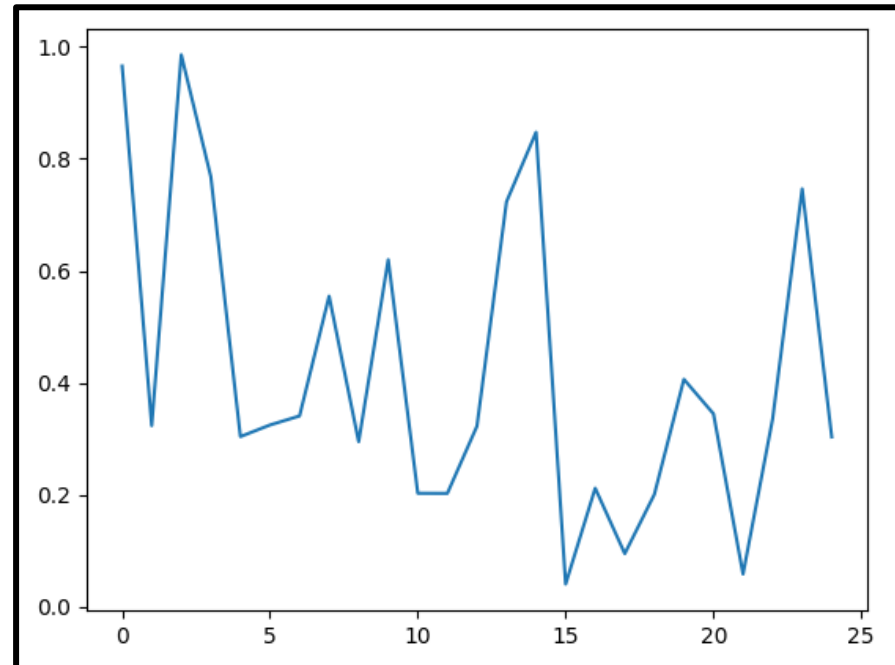
- Wie viele Messungen wurden insgesamt durchgeführt?
- Wie viele Messungen wurden in jedem einzelnen Experiment durchgeführt, deren Messwerte größer als 10 sind?
- Ist der mittlere Messwert von Ada am Radiometer größer als der größte Messwert von Daniel am Thermometer im Experiment_19?
- Was ist der Mittelwert über alle Mittelwerte der einzelnen "Experiment_X.csv" Datensätze?
- Welche Person hat am öftesten das Radiometer benutzt?
- In welchem Experiment wurden die meisten Laser-Messungen im Monat April durchgeführt?



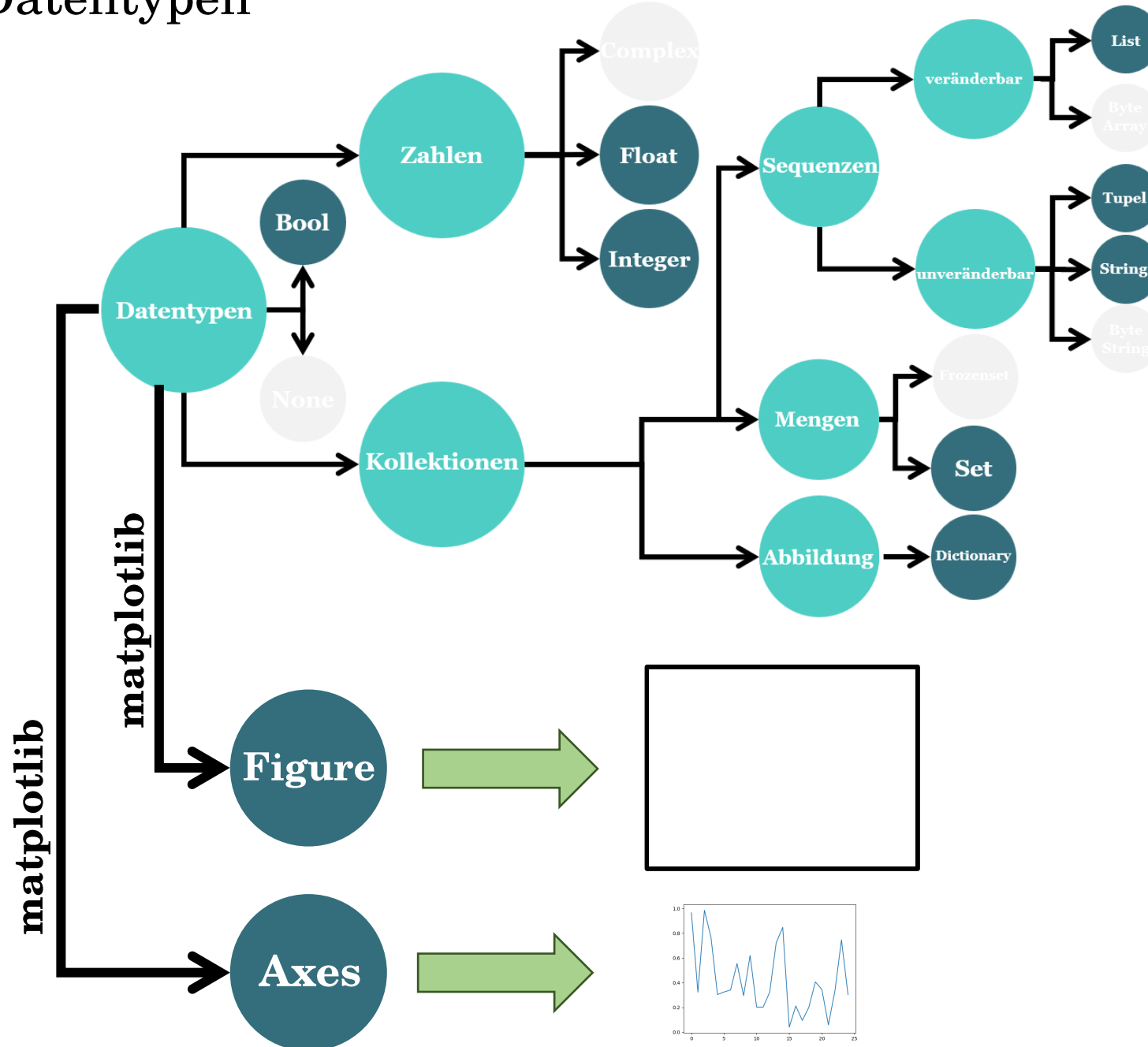
Matplotlib

Das Modul Matplotlib

- **Matplotlib** ist ein Modul welches einem ermöglicht mit nur wenigen Zeilen Code einfach Graphen und Graphiken automatisch zeichnen zu lassen.
- (Und mit vielen Zeilen Code auch sehr aufwändige Graphen und Diagramme).
- Das Modul erweitert Python um den **Figure** und den **Axes** Datentyp, welches einen Graph bzw. ein Diagramme und dessen einzelnen Komponenten darstellt.



Matplotlib Datentypen



Bestandteile des Figure und des Axes Objekts

- Das **Figure** Objekt selbst stellt die "Leinwand" der Grafik dar und kann eines oder mehrere **Axes** Objekte enthalten.
- **Axes** Objekte stellen die eigentlichen Graphen bzw. Diagramme dar und bestehen wiederum aus sehr vielen einzelnen Komponenten:

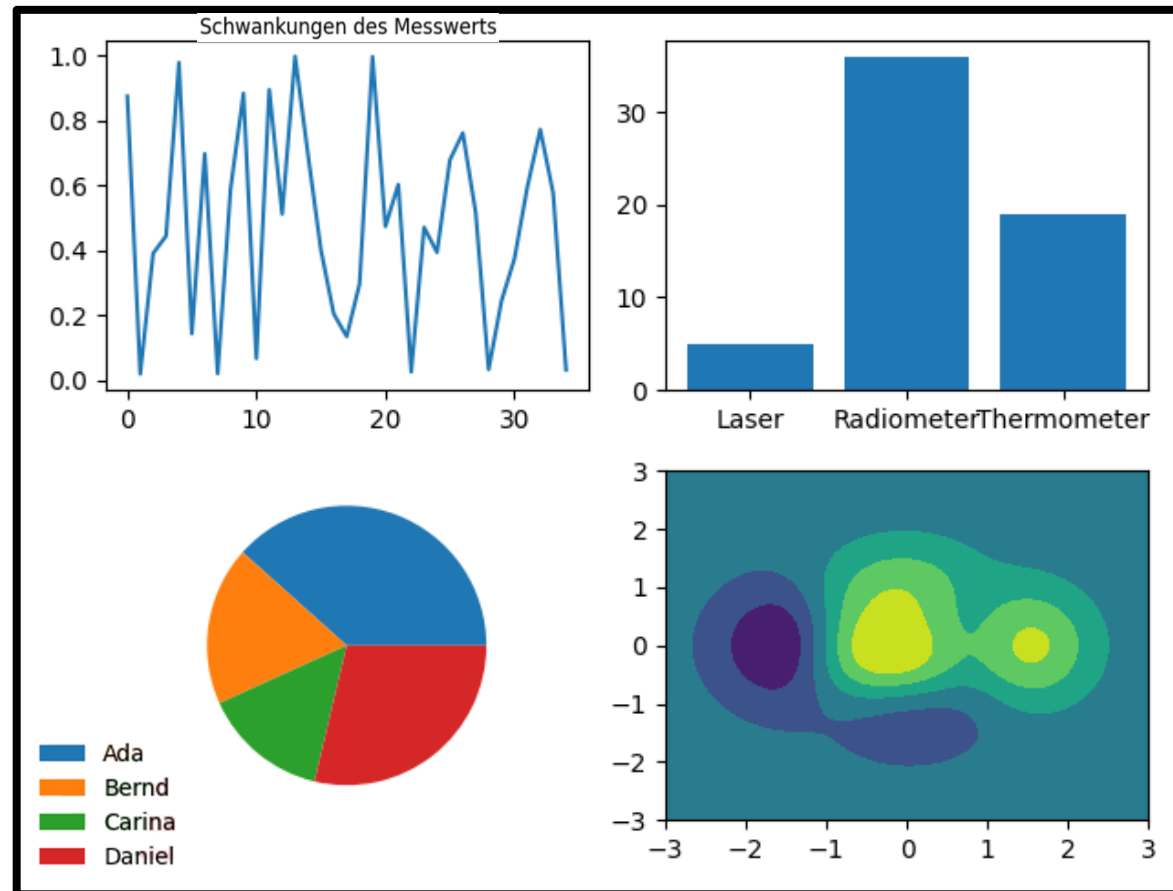


Figure und Axes Objekte erzeugen

- Am leichtesten erzeugt man ein **Figure** Objekt mit einem oder mehreren **Axes** Objekten durch die **subplots** Funktion:
- Zum Speichern des Bildes nutzt man die **savefig** Typfunktion.

```
import matplotlib.pyplot as plt
```

```
fig, ax = plt.subplots() #Ohne Argument erzeugt subplots nur ein Axes Objekt  
fig.savefig('C:/Users/Emanuel/leer.png')
```

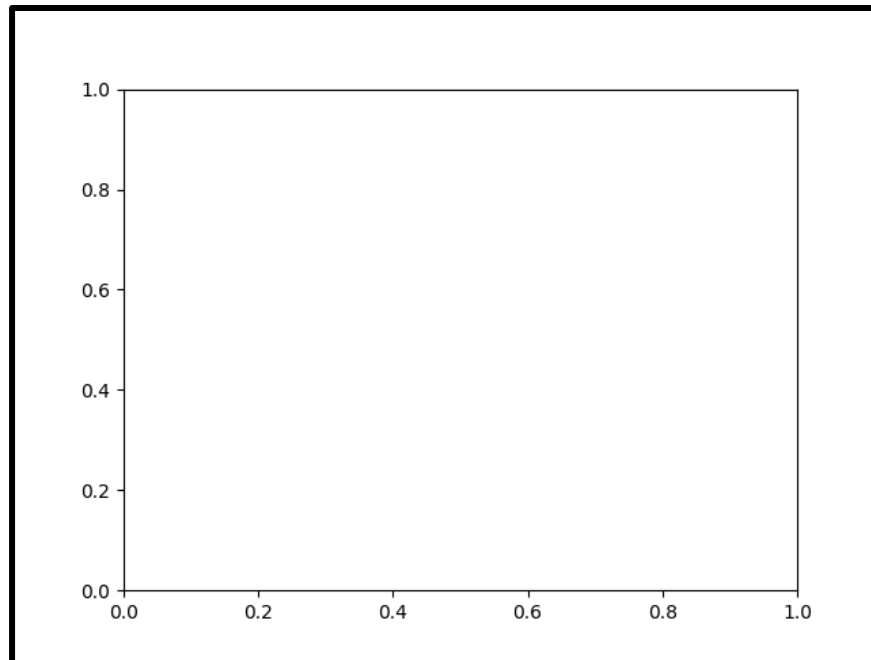
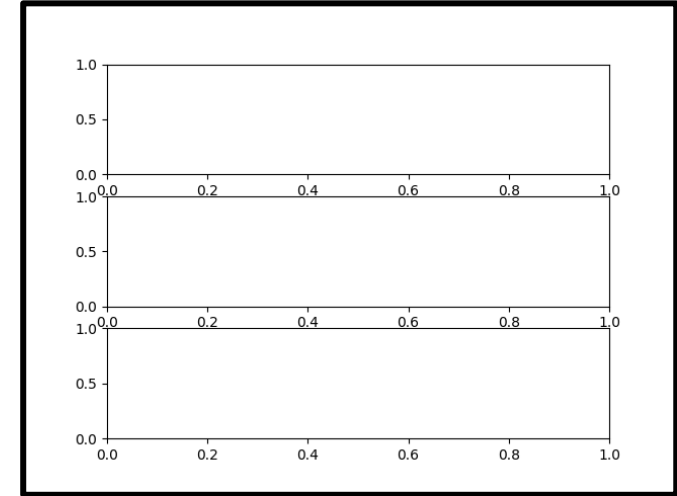


Figure und Axes Objekte erzeugen

- Möchte man ein **Figure** Objekt mit mehreren **Axes** Objekten, so übergibt man der **subplots** Funktion einen Integer als Argument:

```
import matplotlib.pyplot as plt

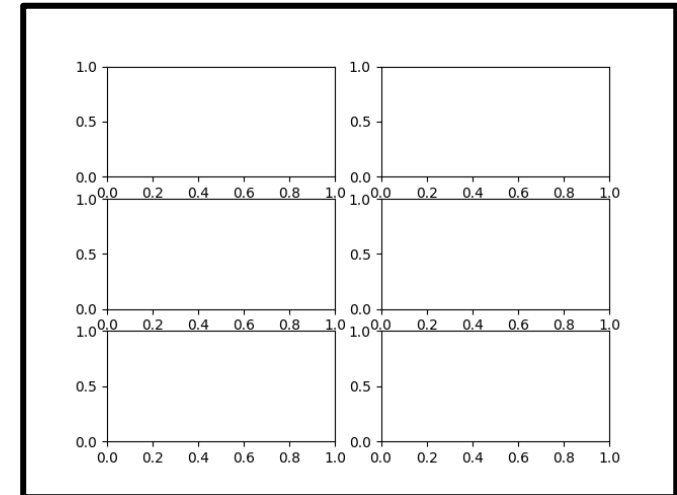
fig, ax = plt.subplots(3)
fig.savefig('C:/Users/Emanuel/leer.png')
print(ax)
[<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]
```



- oder zwei Integer als Argumente:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(3, 2)
fig.savefig('C:/Users/Emanuel/leer.png')
print(ax)
[[<AxesSubplot:> <AxesSubplot:>]
 [<AxesSubplot:> <AxesSubplot:>]
 [<AxesSubplot:> <AxesSubplot:>]]
```



- Als Rückgabe erhält man ein 1- bzw. 2-dimensionales NumPy Array, dessen Elemente die **Axes** Objekte innerhalb des **Figure** Objekts darstellen.

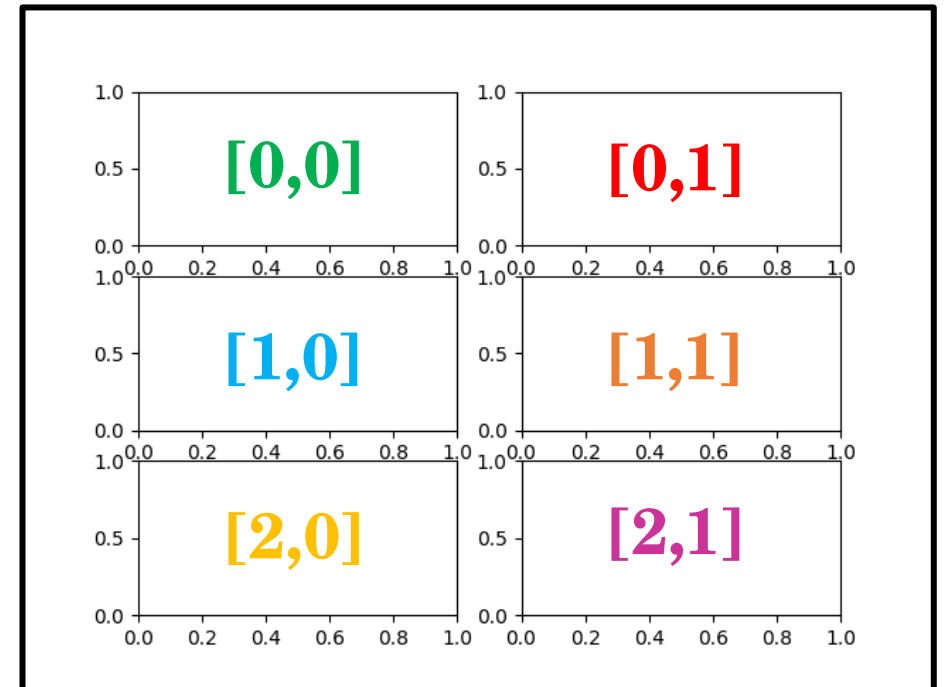
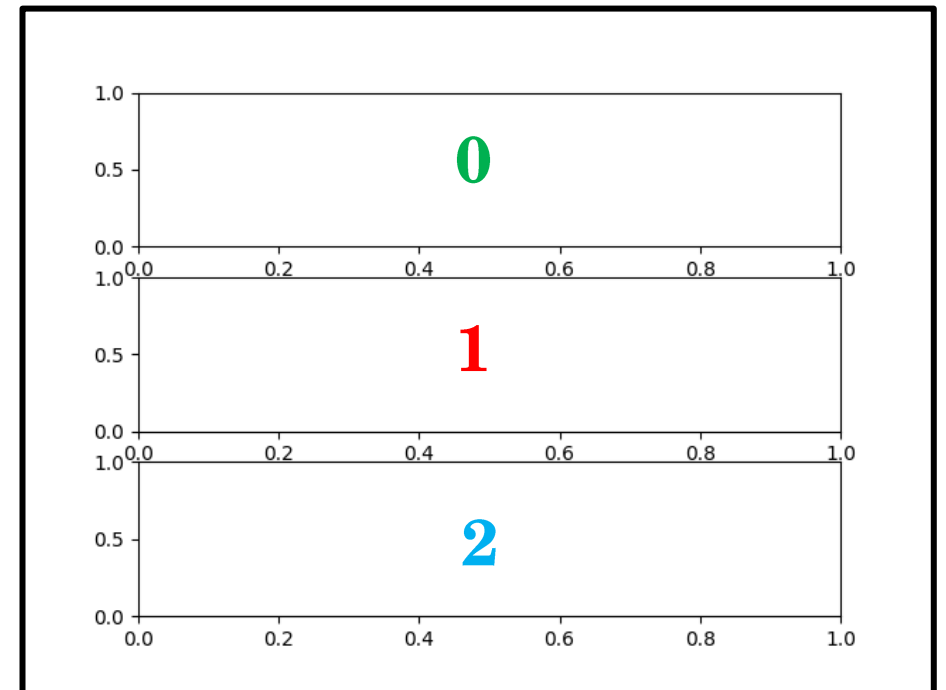
Figure und Axes Objekte erzeugen

```
fig, ax = plt.subplots(3)
fig.savefig('C:/Users/Emanuel/leer.png')
print(ax)
[<AxesSubplot:~>, <AxesSubplot:~>, <AxesSubplot:~>]
```

[0 1 2]

```
fig, ax = plt.subplots(3, 2)
print(ax)
[[<AxesSubplot:~> <AxesSubplot:~>]
 [ <AxesSubplot:~> <AxesSubplot:~>]
 [ <AxesSubplot:~> <AxesSubplot:~>]]
```

[[0,0] [0,1]
[1,0] [1,1]
[2,0] [2,1]]

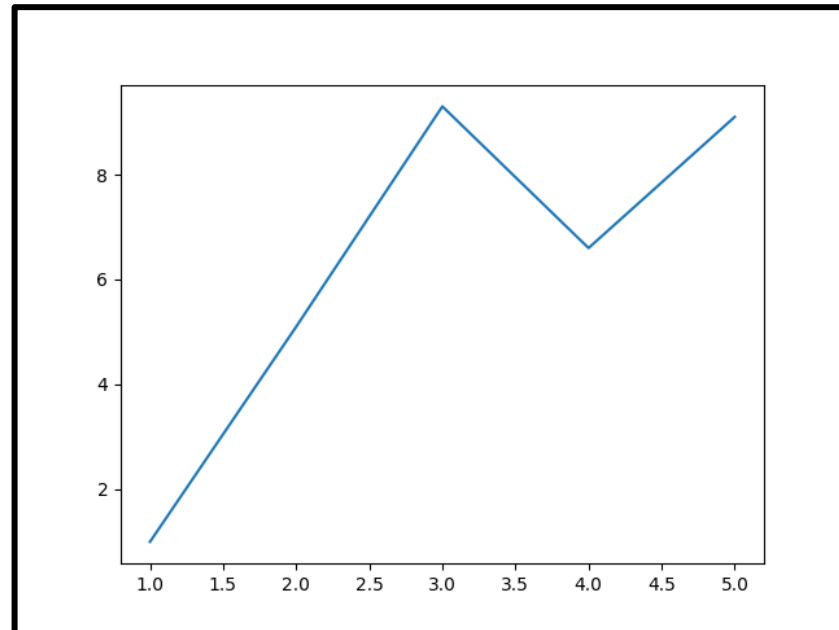


Mit Axes Objekten Graphen zeichnen

- Ein Axes Objekt besitzt (sehr) verschiedene Typfunktionen um Graphen und Diagramme zu zeichnen.
- Das einfachste Diagramm wird mit der `plot` Funktion erzeugt und stellt Punkte in einem X-/Y-Koordinatensystem dar, welche durch eine Linie miteinander verbunden werden:

```
import matplotlib.pyplot as plt

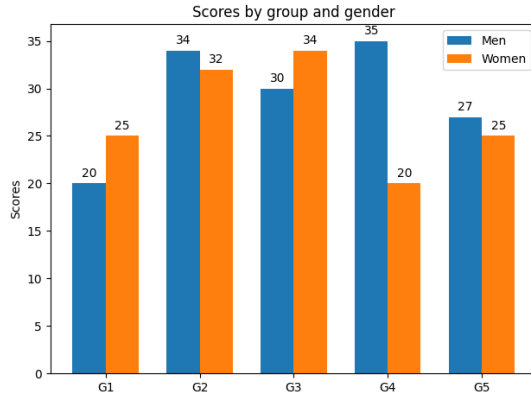
fig, ax = plt.subplots()
ax.plot([1, 2, 3, 4, 5], [1, 5.1, 9.3, 6.6, 9.1])
fig.savefig('C:/Users/Emanuel/plot.png')
```



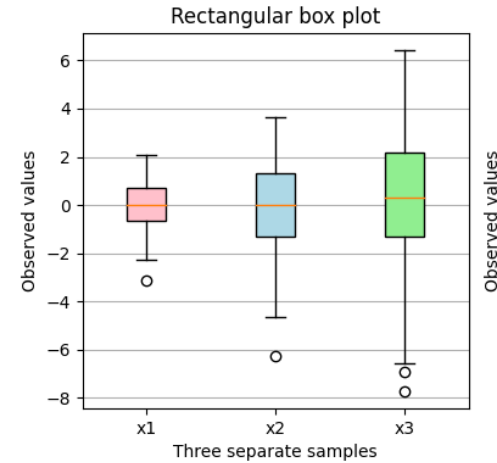
Mit Axes Objekten Graphen zeichnen

- Hier einige Beispiele von Axes Typfunktionen und welche Art von Diagrammen diese erzeugen:

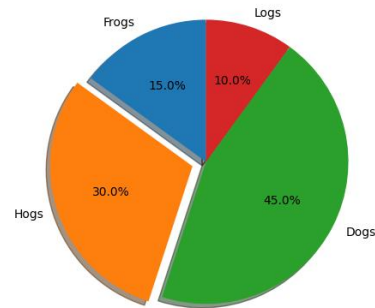
`ax.bar()`



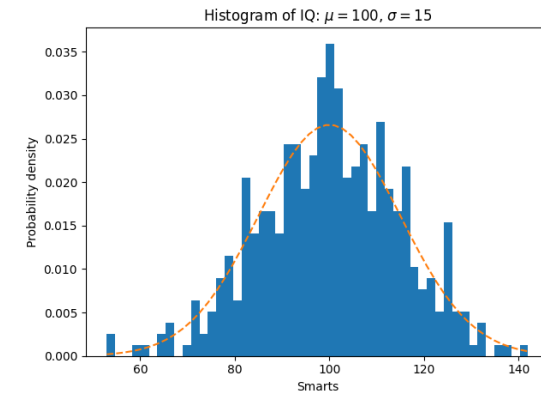
`ax.boxplot()`



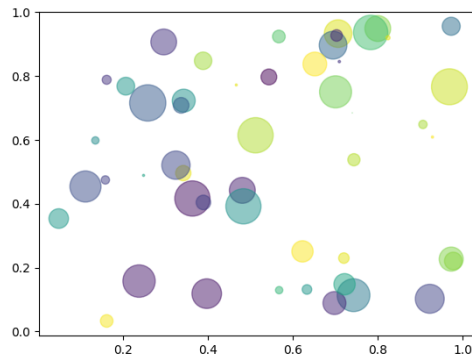
`ax.pie()`



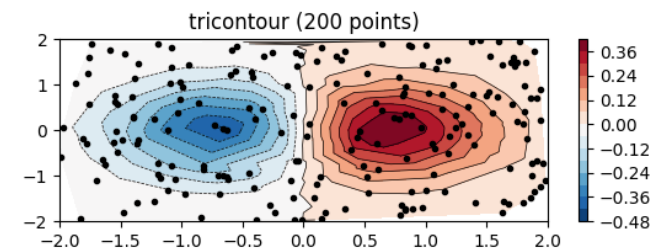
`ax.hist()`



`ax.scatter()`



`ax.contour()`



Mit Axes Objekten Graphen zeichnen

- Da jede Axes Typfunktionen eine andere Art von Diagramm erzeugt und alle diese Diagrammartent aus anderen Elementen bestehen wird auch jedes Mal eine andere Art von Eingabedaten benötigen um die entsprechenden Diagramme zu erzeugen.
- Das heißt leider auch, das diese Typfunktionen nicht einheitlich aufgebaut sind und jeweils ganz eigene Argumente besitzen.
→ hier hilft es viel Gebrauch der help Funktion zu machen und die (sehr gute) offizielle Dokumentation zu befragen (<https://matplotlib.org/stable/index.html>):

```
import matplotlib.pyplot as plt

help(plt.bar)

Help on function bar in module matplotlib.pyplot:

bar(x, height, width=0.8, bottom=None, *, align='center',
    data=None, **kwargs)
    Make a bar plot.

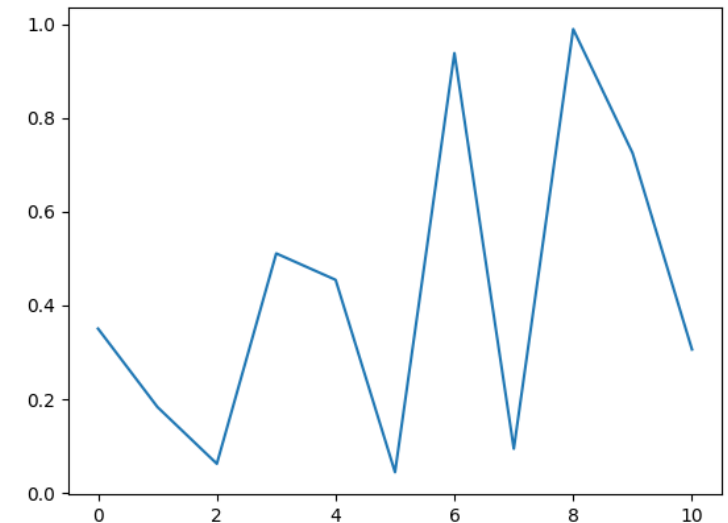
    The bars are positioned at *x* with the given *align*\ment.
    Their
    dimensions are given by *height* and *width*. The vertical
    baseline
    is *bottom* (default 0).
```

Die plot Typfunktion zum Zeichnen von Liniendiagrammen

- Die plot Typfunktion zeichnet einzelne Punkte in einem X-Y-Koordinatensystem und verbindet diese Punkte anschließend durch eine Linie miteinander.
- Diese Art von Diagramm eignet sich zum Beispiel zur Darstellung von Zeitreihendaten.
- Die zwei ersten Argumente welche die plot Funktion als Eingabe erwartet, sind die jeweiligen X- bzw. Y-Koordinaten der Punkte, welche in das Diagramm gezeichnet werden sollen:

```
import matplotlib.pyplot as plt
import numpy as np

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.random(11)           #Erzeugen von 11 Zufallszahlen
fig, ax = plt.subplots()
ax.plot(x, y)
fig.savefig('C:/Users/Emanuel/plot.png')
```

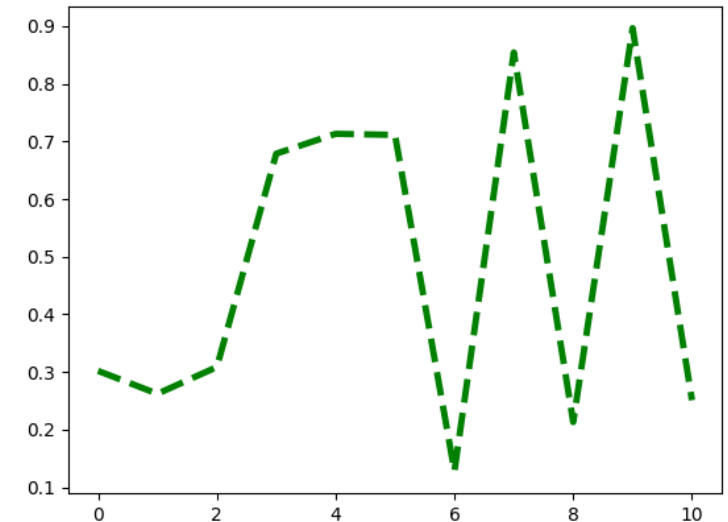


Die plot Typfunktion zum Zeichnen von Liniendiagrammen

- Mit weiteren Argumenten kann man beeinflussen, wie die gezeichneten Elemente des Diagramms dargestellt werden sollen.
- Man kann z. B. die Darstellung der Verbindungslinie durch die optionalen Argumenten `color`, `linewidth`, `linestyle` verändern:

```
import matplotlib.pyplot as plt
import numpy as np

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.random(11)           #Erzeugen von 11 zufallszahlen
fig, ax = plt.subplots()
ax.plot(x, y, color='green', linewidth=3.5, linestyle='--')
fig.savefig('C:/Users/Emanuel/plot.png')
```

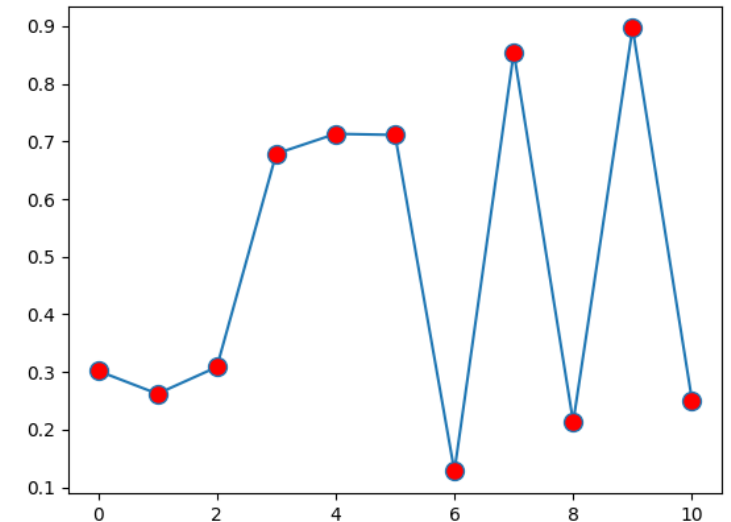


Die plot Typfunktion zum Zeichnen von Liniendiagrammen

- Mit weiteren Argumenten kann man beeinflussen, wie die gezeichneten Elemente des Diagramms dargestellt werden sollen.
- Man kann z. B. die Darstellung der X-Y-Punkte durch die optionalen Argumenten **marker**, **markersize**, **mfc** verändern:

```
import matplotlib.pyplot as plt
import numpy as np

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.random(11)          #Erzeugen von 11 zufallszahlen
fig, ax = plt.subplots()
ax.plot(x, y, marker='o', markersize=9, mfc='red')
fig.savefig('C:/Users/Emanuel/plot.png')
```

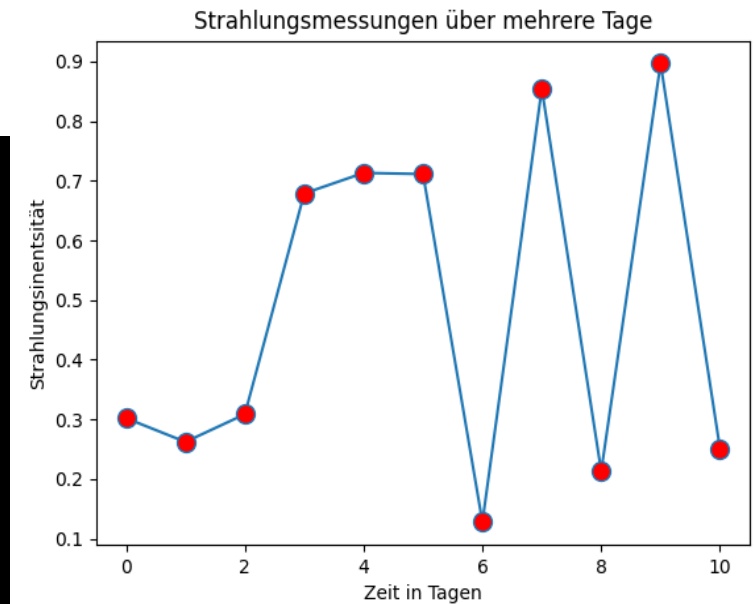


Die plot Typfunktion zum Zeichnen von Liniendiagrammen

- Mit der Funktion `set_title` lässt sich jedem Diagramm ein Titel hinzufügen.
- Die X- bzw. Y-Achse lässt sich mit der Funktion `set_xlabel` bzw. `set_ylabel` beschriften.

```
import matplotlib.pyplot as plt
import numpy as np

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.random(11)          #Erzeugen von 11 zufallszahlen
fig, ax = plt.subplots()
ax.plot(x, y, marker='o', markersize=9, mfc='red')
ax.set_title('Strahlungsmessungen über mehrere Tage')
ax.set_xlabel('Zeit in Tagen')
ax.set_ylabel('Strahlungsinentsität')
fig.savefig('C:/Users/Emanuel/plot.png')
```

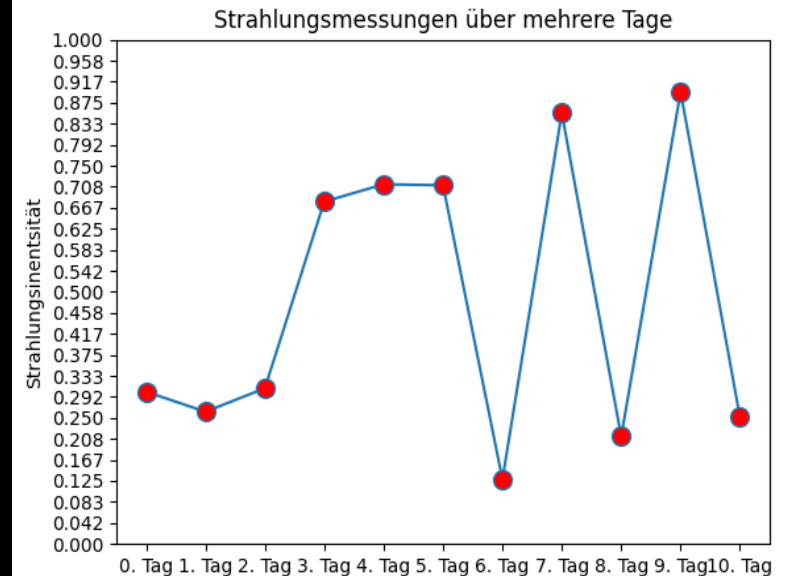


Die plot Typfunktion zum Zeichnen von Liniendiagrammen

- Man kann auch direkt die Anzahl und die Beschriftung der einzelnen Markierungen auf den Skalen der X- und der Y-Achse mittels der folgenden Typfunktionen beeinflussen: `set_xticks`, `set_yticks`, `set_xticklabels`, `set_yticklabels`:

```
import matplotlib.pyplot as plt
import numpy as np

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.random(11)          #Erzeugen von 11 zufallszahlen
fig, ax = plt.subplots()
ax.plot(x, y, marker='o', markersize=9, mfc='red')
ax.set_title('Strahlungsmessungen über mehrere Tage')
ax.set_xticks(x)
ax.set_xticklabels([f'{i}. Tag' for i in x])
ax.set_ylabel('Strahlungsinentsität')
ax.set_yticks(np.linspace(0,1,25))
ax.set_ylabel('Strahlungsinentsität')
fig.savefig('C:/Users/Emanuel/plot.png')
```

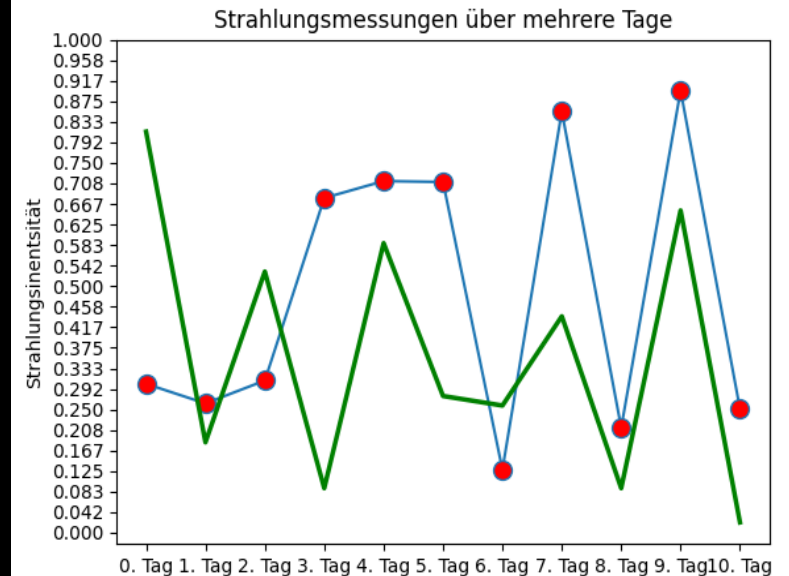


Die plot Typfunktion zum Zeichnen von Liniendiagrammen

- Innerhalb eines **Axes** Objekts, lassen sich auch direkt mehrere Diagramme vom selben Diagrammtyp darstellen, in dem man die entsprechende Typfunktion zum Zeichnen des Diagramm nocheinmal (mit anderen Eingabeargumenten) wiederholt.
- Hier ein Beispiel mit zwei Liniendiagrammen innerhalb eines **Axes** Objekts:

```
import matplotlib.pyplot as plt
import numpy as np

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.random(11)           #Erzeugen von 11 zufallszahlen
z = np.random.random(11)           #Erzeugen von 11 zufallszahlen
fig, ax = plt.subplots()
ax.plot(x, y, marker='o', markersize=9, mfc='red')
ax.plot(x, z, color='green', linewidth=2.5)
ax.set_title('Strahlungsmessungen über mehrere Tage')
ax.set_xticks(x)
ax.set_xticklabels([f'{i}. Tag' for i in x])
ax.set_ylabel('Strahlungsinentsität')
ax.set_yticks(np.linspace(0,1,25))
ax.set_ylabel('Strahlungsinentsität')
fig.savefig('C:/Users/Emanuel/plot.png')
```

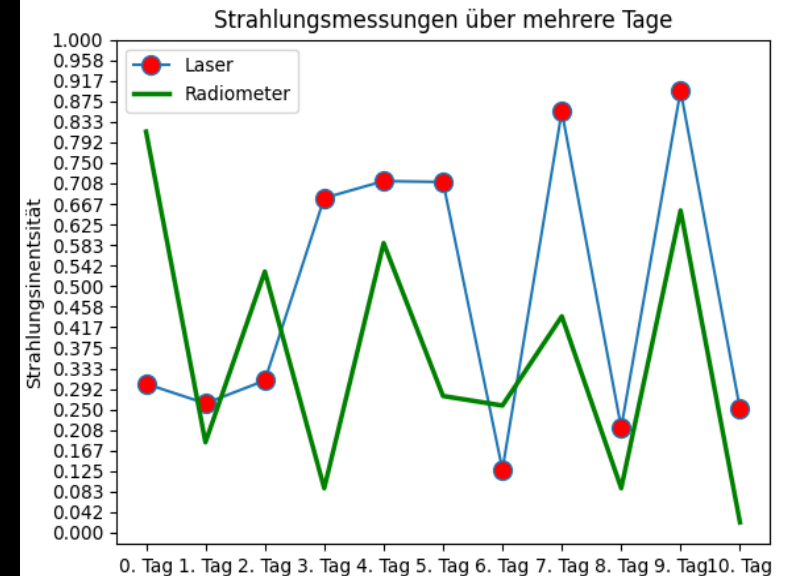


Die plot Typfunktion zum Zeichnen von Liniendiagrammen

- Mittels des `label` Arguments, kann man den einzelnen Liniendiagrammen auch Namen geben.
- Mit Hilfe dieser Namen, kann die `legend` Typfunktion automatische eine Legende zum Diagramm erzeugen:

```
import matplotlib.pyplot as plt
import numpy as np

x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.random(11)           #Erzeugen von 11 Zufallszahlen
z = np.random.random(11)           #Erzeugen von 11 Zufallszahlen
fig, ax = plt.subplots()
ax.plot(x, y, marker='o', markersize=9, mfc='red', label='Laser')
ax.plot(x, z, color='green', linewidth=2.5, label='Radiometer')
ax.set_title('Strahlungsmessungen über mehrere Tage')
ax.set_xticks(x)
ax.set_xticklabels([f'{i}. Tag' for i in x])
ax.set_ylabel('Strahlungsinentsität')
ax.set_yticks(np.linspace(0,1,25))
ax.set_ylabel('Strahlungsinentsität')
ax.legend()
fig.savefig('C:/Users/Emanuel/plot.png')
```

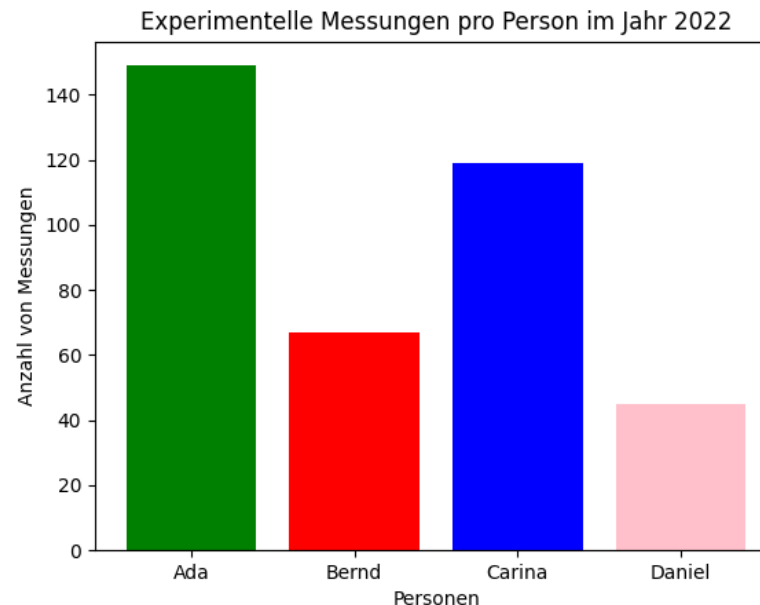


Die bar Typfunktion zum Zeichnen von Balkendiagrammen

```
import matplotlib.pyplot as plt

personen = ('Ada', 'Bernd', 'Carina', 'Daniel')
anzahlMessungen = [149, 67, 119, 45]
farben = ('green', 'red', 'blue', 'pink')

fig, ax = plt.subplots()
ax.bar(personen, anzahlMessungen, color=farben)
ax.set_title('Experimentelle Messungen pro Person im Jahr 2022')
ax.set_xlabel('Personen')
ax.set_ylabel('Anzahl von Messungen')
fig.savefig('C:/Users/Emanuel/plot.png')
```



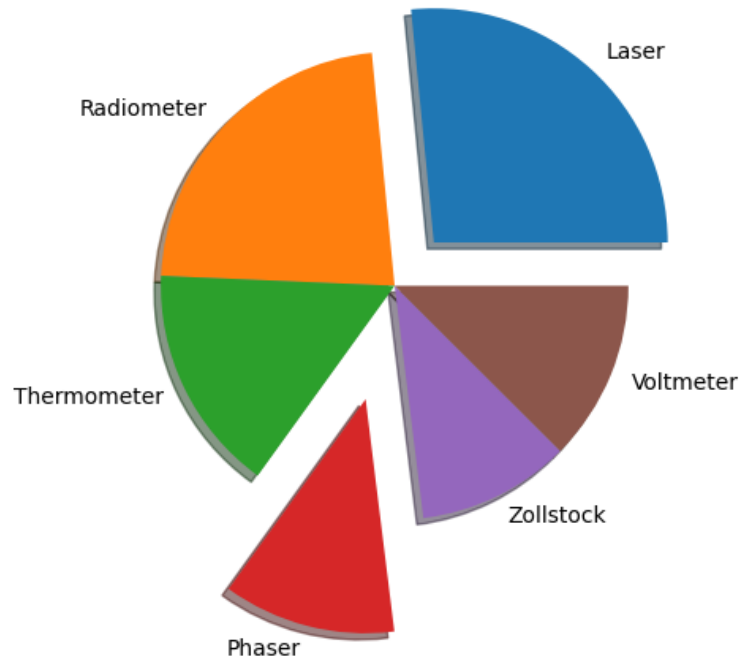
Die pie Typfunktion zum Zeichnen von Kreisdiagrammen

```
import matplotlib.pyplot as plt

messtechniken = ('Laser', 'Radiometer', 'Thermometer', 'Phaser', 'Zollstock', 'Voltmeter')
häufigkeit=[98, 84, 58, 44, 39, 46]

fig, ax = plt.subplots()
ax.pie(häufigkeit, labels=messtechniken, explode=[0.25,0,0,0.5,0,0], shadow=True)
ax.set_title('Anteil der verschiedenen Messgeräte an den Messungen')
fig.savefig('C:/Users/Emanuel/plot.png')
```

Anteil der verschiedenen Messgeräte an den Messungen

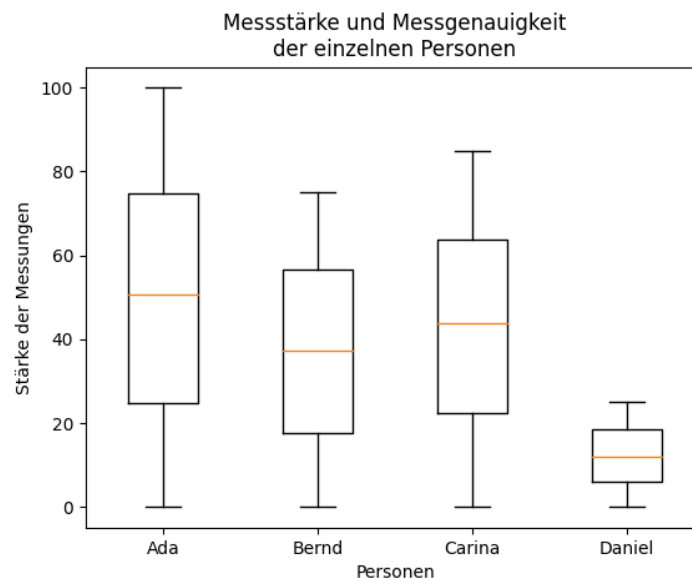


Die boxplot Typfunktion zum Zeichnen von Box-Whisker-Diagrammen

```
import matplotlib.pyplot as plt

personen = ('Ada', 'Bernd', 'Carina', 'Daniel')
messwerte = [np.random.random(1000)*100,
              np.random.random(1000)*75,
              np.random.random(1000)*85,
              np.random.random(1000)*25]

fig, ax = plt.subplots()
ax.boxplot(messwerte)
ax.set_title('Messstärke und Messgenauigkeit\nder einzelnen Personen')
ax.set_xlabel('Personen')
ax.set_xticklabels(personen)
ax.set_ylabel('Stärke der Messungen')
fig.savefig('C:/Users/Emanuel/plot.png')
```

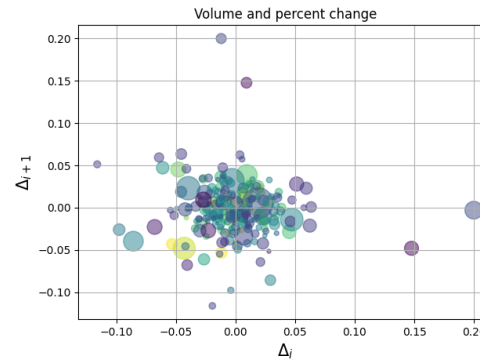


Die wunderbare Welt von Matplotlib

- Die offizielle Seite von Matplotlib bietet nicht nur eine sehr ausführliche und gut geschriebene Dokumentation aller Diagrammart, Typfunktionen inklusiver Argumente, sondern eine große Galerie an Beispiel Diagrammen mit dem dazugehörigem Beispielcode.
- So lassen sich auch sehr komplexe Grafiken und Diagramme einfach nachbauen und nachvollziehen (<https://matplotlib.org/stable/gallery/>):

Scatter Demo2

Demo of scatter plot with varying marker colors and sizes.



```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook

# Load a numpy record array from yahoo csv data with fields date, open, close,
# volume, adj_close from the mpl-data/example directory. The record array
# stores the data as an np.datetime64 with a day unit ('D') in the date column.
price_data = (cbook.get_sample_data('goog.npy', np_load=True)['price_data']
               .view(np.recarray))
price_data = price_data[-250:] # get the most recent 250 trading days

deltai = np.diff(price_data.adj_close) / price_data.adj_close[:-1]

# Marker size in units of points^2
volume = (15 * price_data.volume[:-2] / price_data.volume[0])**2
close = 0.003 * price_data.close[:-2] / 0.003 * price_data.open[:-2]

fig, ax = plt.subplots()
ax.scatter(deltai[:-1], deltai[1:], c=close, s=volume, alpha=0.5)

ax.set_xlabel(r'$\Delta_i$', fontsize=15)
ax.set_ylabel(r'$\Delta_{i+1}$', fontsize=15)
ax.set_title('Volume and percent change')

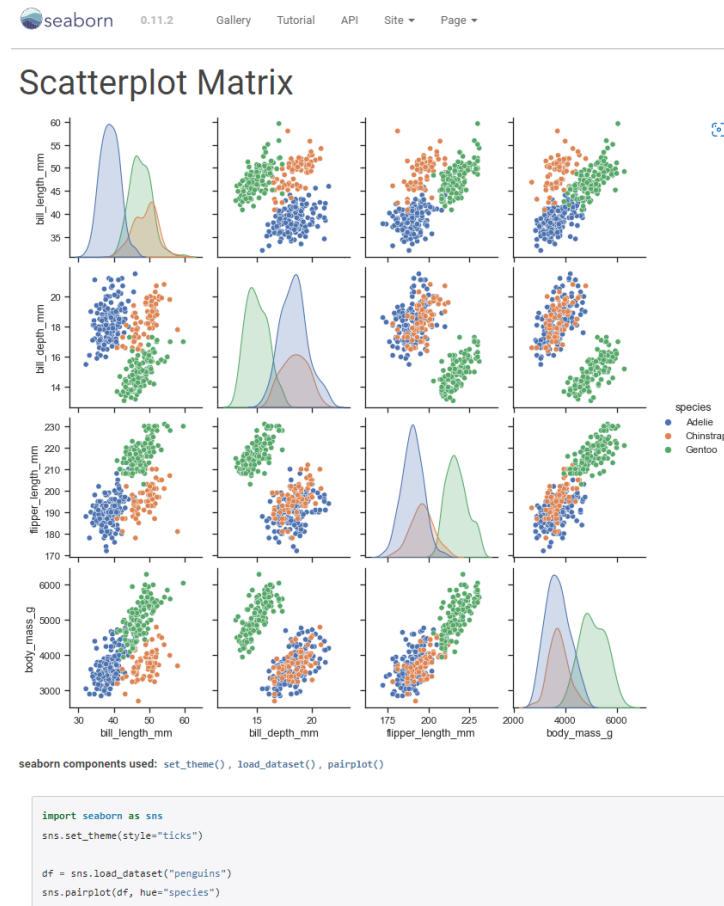
ax.grid(True)
fig.tight_layout()

plt.show()
```

[Download Python source code: scatter_demo2.py](#)

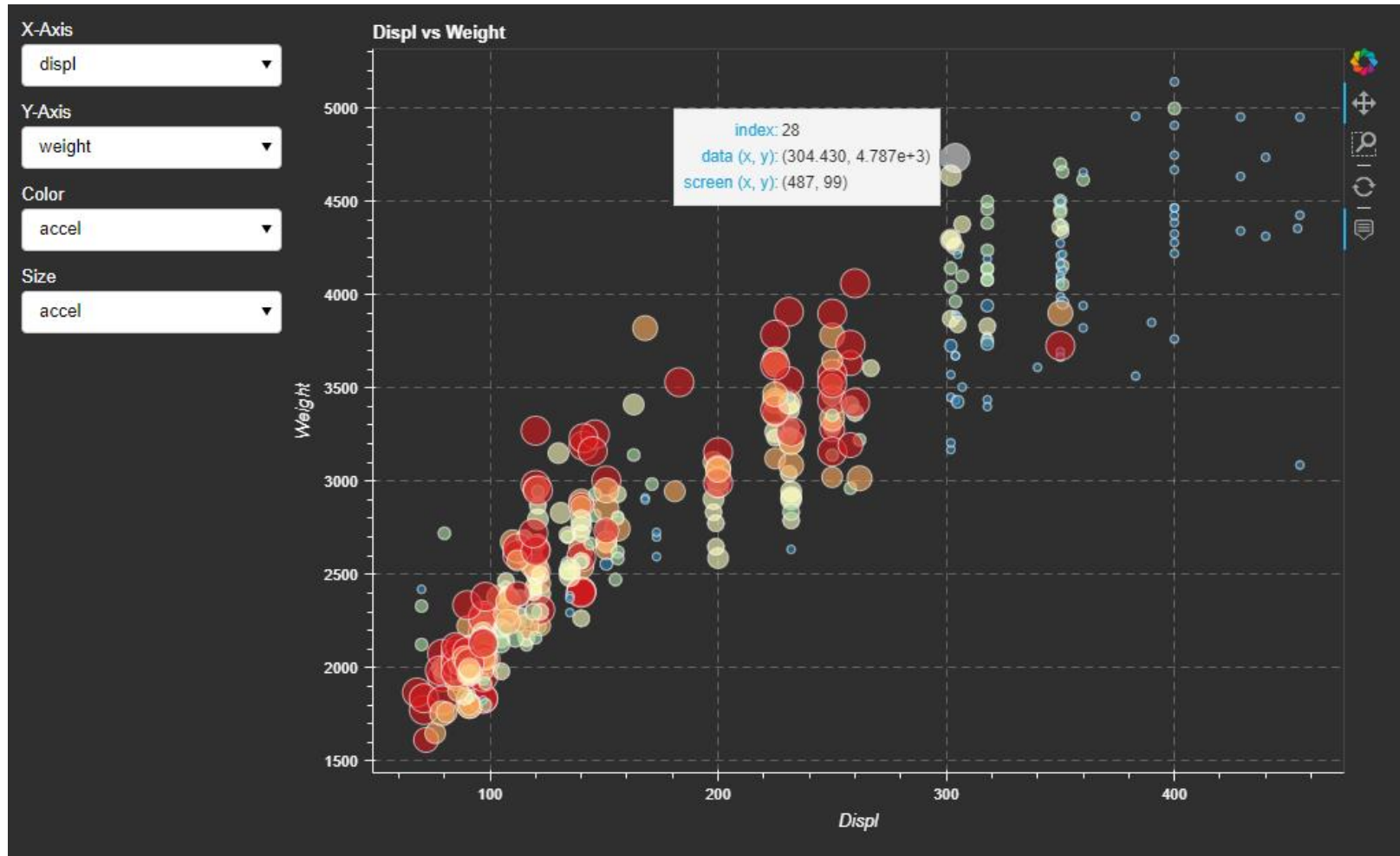
Und noch weiter über Matplotlib hinaus

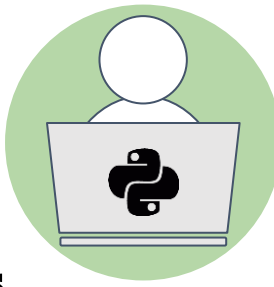
- Es gibt noch Module die aufbauend auf Matplotlib die Erstellung von Graphen und Diagrammen erweitern (und teilweise auch erleichtern).
- Das prominenteste Beispiel ist das Modul Seaborn: <https://seaborn.pydata.org/>



Und noch weiter über Matplotlib hinaus

- Es lassen sich aber auch interaktive Diagramme mit dem Modul Bokeh erstellen:
<https://bokeh.org/>





Aufgabe:

Schreibt euch ein Programm, welches zu einer beliebigen der "Experiment_X.csv" Dateien aus eurem "beispieldaten" Ordner die folgenden Diagramme mit Hilfe von **Matplotlib** erzeugt und als Bilddatei im "beispieldaten" Ordner speichert:

- Ein Balkendiagramm, welches die Anzahl der Lasermessungen für jeden der 12 Monate darstellt.
- Ein Box-Whiskers-Diagramm in denen die Messwerte aller vier Personen am Radiometer miteinander verglichen werden können.
- Ein Kreisdiagramm, welches die Gesamtzahl aller Messungen jeder der vier Personen im Verhältnis zueinander darstellt.

Wenn euer Programm funktioniert erweitert es so, dass es diese drei Diagramme automatisch für alle "Experiment_X.csv" Dateien nacheinander erzeugt. Am Ende soll es also für jede "Experiment_X.csv" Datei genau drei Bilddateien geben, welche die oben genannte Diagramme des jeweiligen Datensatzes darstellen.