

# Python Module

November 14, 2024

## 1 Module in Python

Vorlesung Theoretische Chemie (WiSe 2024/2025)

## 2 Was ist ein Python Modul

- **Modul:** Erweiterung der Python Sprache, die spezielle Funktionen / Konstanten bereitstellt
- ständige Neuimplementierung von Basisfunktionen ist nicht notwendig

**Bsp. Konstanten und Mathematische Funktionen:**

```
[5]: import numpy as np
      print(np.pi)      # Kreiszahl Pi
      print(np.sin(0))  # Sinusfunktion
```

3.141592653589793

0.0

## 3 Relevante Module für Chemie / Wissenschaften

- **NumPy:** mathematische Funktionen, Lineare Algebra, Daten Einlesen
- **Matplotlib:** Graphen erstellen
- **SciPy:** Regression, Optimierungsprobleme, ...
- **Pandas:** Analyse von großen Datensätzen
- ...

## 4 Installation von Modulen

- nicht alle Module sind in der Installation von Python enthalten
- Installation abhängig von Paketmanager
  - **pip** (standard Python Paketmanager) `bash` `pip install numpy`
  - **conda** Anaconda Paketmanager `bash` `conda install numpy`
- Python Editoren wie PyCharm ermöglichen auch das Installieren mit einer grafischen Benutzeroberfläche

## 5 Einbinden von Python Modulen

Einfacher Import:

```
import modul
```

Import mit alias:

```
import modul as alias      # bevorzugt
```

Import von spezifischen Funktionen/Objekten:

```
from modul import module_function
```

```
[6]: import numpy
      print(numpy.pi)
      print(numpy.sin(0))
```

```
3.141592653589793
0.0
```

```
[7]: import numpy as np
      print(np.pi)
      print(np.sin(0))
```

```
3.141592653589793
0.0
```

```
[8]: from numpy import pi, sin
      print(pi)
      print(sin(0))
```

```
3.141592653589793
0.0
```

## 6 Wo Modul import statement setzen

- als allererstes im Python Script
- jedes Modul nur einmal importieren

```
# als erstes Module importieren
```

```
import module1 as m1
```

```
import module2
```

```
from module3 import function_5
```

```
# danach folgen Funktionsdefinitionen
```

```
def my_function1():
```

```
    ...
```

```
    return ...
```

```
# danach die eigentlichen Berechnungen
```

```
a = ...
```

```
b = ...
```

```
print(...)
```

## 7 Numpy

## 8 Numpy: Überblick

- Numpy: “Numerical Python”
- **Inhalt**
  - Konstanten: Eulersche Zahl, Kreiszahl  $\pi$ , ...
  - Funktionen: trigonometrische Funktionen, exponential und logarithmus Funktionen, numerische Integration, numerische Ableitung, ...
  - **lineare Algebra**: Matrizen (Tensoren), Matrix Operationen (Eigenwert Löser, Matrix Produkt, ...)
  - einlesen von Datensätzen

## 9 Numpy: Arrays (Matrizen, Tensoren)

- Arrays bezeichnet den Datentyp zum Speichern eines Tensors

Array-Typ	Bezeichnung
0D-Array	Skalar
1D-Array	Vektor
2D-Array	Matrix
3D-Array	Rang-3 Tensor
nD-Array	Rang-n Tensor

- Elemente im Tensor haben einen einzigen Datentypen (z.B. Float oder Integer)

<h3>Brauch ich Arrays auch wenn man als Experimentalchemiker arbeite?</h3>

<p><b>Ja</b>: Arbeiten mit Datensätzen, z.B. Spektren mit x und y listen als 1D Arrays, oder 2D

## 10 Numpy: Erstellen eines Arrays

### 10.1 aus Listen

```
[9]: import numpy as np
my_list = [1, 2, 3, 4]
my_array = np.array(my_list)      # erstellt den 1D-Array
print(my_array)

my_array = np.array([1, 2, 3, 4]) # auch so möglich
print(my_array)
```

```
[1 2 3 4]
```

```
[1 2 3 4]
```

```
[10]: import numpy as np
my_array = np.array([[1, 2], [3, 4], [5, 6]]) # erstellt eine Matrix
print(my_array)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

## 11 Numpy: Erstellen eines Arrays

### 11.1 mit generierenden Funktionen

```
[11]: import numpy as np
my_array = np.linspace(-1.0, 2.0, 10) # 1D-Array: (start, stop, elements)
print(my_array)
my_array = np.arange(-1.0, 2.0, 0.3) # 1D-Array: (start, stop, step)
print(my_array)
```

```
[-1.          -0.66666667 -0.33333333  0.           0.33333333  0.66666667
  1.           1.33333333  1.66666667  2.           ]
[-1.  -0.7 -0.4 -0.1  0.2  0.5  0.8  1.1  1.4  1.7]
```

```
[12]: import numpy as np
my_array = np.zeros((2, 3)) # 2D-Array: (dim1, dim2)
print(my_array)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

## 12 Numpy: Python Listen vs Numpy Arrays

```
[51]: import numpy as np
my_array = np.array([1, 2, 3, 4])
result = my_array * 3 + 1 # Operation wird für jedes Element ausgeführt
print(result)

result = np.sin(my_array)
print(result)
```

```
[ 4  7 10 13]
[ 0.84147098  0.90929743  0.14112001 -0.7568025 ]
```

```
[14]: my_list = [1, 2, 3, 4]
result = my_list * 3 + 1 # Mit Listen geht das nicht
print(result)

#my_list = [[1, 2], [3, 4]]
#my_list[0, 1]
```

-----  
TypeError

Traceback (most recent call last)

```
Cell In[14], line 2
      1 my_list = [1, 2, 3, 4]
----> 2 result = my_list * 3 + 1 # Mit Listen geht das nicht
      3 print(result)
```

**TypeError:** can only concatenate list (not "int") to list

## 13 Numpy: Array Indexierung

- Zugreifen auf ein bestimmtes Element in einem Array
- index in **eckigen Klammern**: `numpy_array[index]`
- Simpel für **1D Arrays**:

```
[15]: import numpy as np
my_array = np.array([1, 2, 3])
print(my_array[0]) # erstes Element
print(my_array[1]) # mittleres Element
print(my_array[2]) # letztes Element
print(my_array[-1]) # auch letztes Element
```

1  
2  
3  
3

**Wichtig**

Das erste Element in einem Array hat den Index 0.

## 14 Numpy: Array Indexierung

- Für **2D Arrays**: `my_2d_array[index_axis_0, index_axis_1]`

```
[16]: import numpy as np
my_array = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
print(my_array[0, 0]) # element links oben
print(my_array[2, 2]) # element rechts unten
print(my_array[1, 0]) # 2. element in der ersten Spalte
```

1  
9  
4

## 15 Numpy: Array Indexierung

- Ausschnitt eines Arrays mit mehr als einem Element:

```
my_1D_array[start_index : end_index] # Ausschnitt mit ":" operator
```

```
[52]: import numpy as np
my_array = np.array([1, 2, 3, 4, 5])
print(my_array[0:2])
print(my_array[2:]) # weglassen eines index bedeutet alles danach
print(my_array[:2]) # oder davor
print(my_array[:])
```

```
[1 2]
[3 4 5]
[1 2]
[1 2 3 4 5]
```

## 16 Numpy: Array Indexierung

- Ausschnitt eines 2D Arrays mit mehr als einem Element:

```
my_2D_array[start_index_0 : end_index_0, start_index_1 : end_index_1]
```

```
[53]: import numpy as np
my_array = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
print(my_array[0:2, 0:2]) # sub-matrix links oben
print(my_array[:, 0]) # erste Spalte
print(my_array[1, :]) # zweite Zeile
```

```
[[1 2]
 [4 5]]
[1 4 7]
[4 5 6]
```

## 17 Numpy: Logische Indexierung

- oft gewünscht: Elemente aus einem Array, die eine Bedingung erfüllen
- Bsp. alle Elemente die größer sind als ein Schwellenwert

```
[19]: import numpy as np
my_array = np.array([1, 2, 3, 4])
mask = [False, False, True, True] # nur die letzten beiden elemente
print(my_array[mask])
```

[3 4]

```
[20]: import numpy as np
my_array = np.array([1, 2, 3, 4])
mask = (my_array > 2.0) # erstellt liste mit True oder False Werten
print(mask)
print(my_array[mask])
```

[False False True True]

[3 4]

```
[21]: import numpy as np
my_array = np.array([1, 2, 3, 4])
print(my_array[my_array > 2.0]) # alles in einem Ausdruck
```

[3 4]

## 18 Numpy: Array Dimensionen

- Die Form eines Arrays erhält man mit der `numpy.shape` Funktion
- Es wird eine Tupel mit den Längen der jeweiligen Dimensionen zurückgegeben

```
[54]: import numpy as np
my_array = np.array([1, 2, 3, 4, 5]) # 1D Array
my_shape = np.shape(my_array)
print(my_shape)
print(f"Der Array hat die Länge {my_shape[0]}.")
```

(5,)

Der Array hat die Länge 5.

```
[55]: import numpy as np
my_array = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])
my_shape = np.shape(my_array)
print(my_shape)
print(f"Der Array hat die Form {my_shape[0]} mal {my_shape[1]}.")
```

(3, 3)

Der Array hat die Form 3 mal 3.

## 19 Numpy: Array aus Datei einlesen

- Mit der Funktion `numpy.loadtxt` ist es möglich Textdateien zu laden

```
my_array = np.loadtxt("my_txt_file.txt")
```

- optionale Argumente helfen beim einlesen

*# einlesen von komma-separierten text file, nur die ersten beiden Spalten*

```
my_array = np.loadtxt("my_txt_file.csv", delimiter=',', usecols=(0, 1))
```

```
[24]: import numpy as np
      """
      Inhalt der Textdatei test.txt:

      1.0 2.0 3.0
      4.0 5.0 6.0
      7.0 8.0 9.0
      """

      my_array = np.loadtxt("test.txt")
      print(my_array)
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

## 20 Numpy: Funktionen und Arrays

- numpy stellt sehr viele mathematische Funktionen bereit
- Beispiele einiger Funktionen:

```
[25]: import numpy as np
      my_array = np.array([-2, -1, 1, 2])

      print(np.sin(my_array))           # trigonometrische Funktionen
      #print(np.sum(my_array))          # Summe aller Elemente
      #print(np.abs(my_array))          # absolut Wert aller Elemente
      #print(np.log(np.abs(my_array)))  # Logarithmus
      #print(np.amax(my_array))         # Maximum aller Elementen
      #print(np.amin(my_array))         # Minimum aller Elemente
      #print(np.mean(my_array))         # Mittelwert
      #print(np.trapz(my_array))        # numerische Integration mit trapez-regel
      #print(np.dot(my_array, my_array)) # Skalarprodukt
```

```
[-0.90929743 -0.84147098  0.84147098  0.90929743]
```



## 21 Numpy: Achtung bei Array Kopien!

```
[56]: import numpy as np
a = np.array([1, 1, 1])
b = a
b[0] = 2
print(a, b)
```

```
[2 1 1] [2 1 1]
```

**Grund:** Python speichert alles als Objekte. Mit `b = a` enthält die variable `b` die Referenz zu dem Objekt der Variable `a`, aber kein neues Objekt wird erzeugt.

**Workaround:**

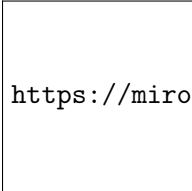
```
import numpy as np
import copy
b = copy.deepcopy(a)
```

## 22 Numpy: Übung

[Link zur Übung](#)

## 23 Matplotlib

## 24 Matplotlib: Overview



[https://miro.medium.com/v2/resize:fit:9450/1\\*0AFEIg9w1XHyZk0xBud14A.png](https://miro.medium.com/v2/resize:fit:9450/1*0AFEIg9w1XHyZk0xBud14A.png)

### 24.1 Matplotlib vs Origin und Excel

- **Nachteile von Matplotlib** gegenüber Origin und Excel:
  - Programmierkenntnisse erforderlich
  - etwas flachere Lernkurve
- **Vorteile von Matplotlib** gegenüber Origin und Excel:
  - Open source (kostenlos nutzbar)
  - mehr Funktionen als in Origin und Excel
  - große Datensätze kein Problem
  - Automatisierung
  - Wiederverwendbarkeit
  - sieht professioneller aus

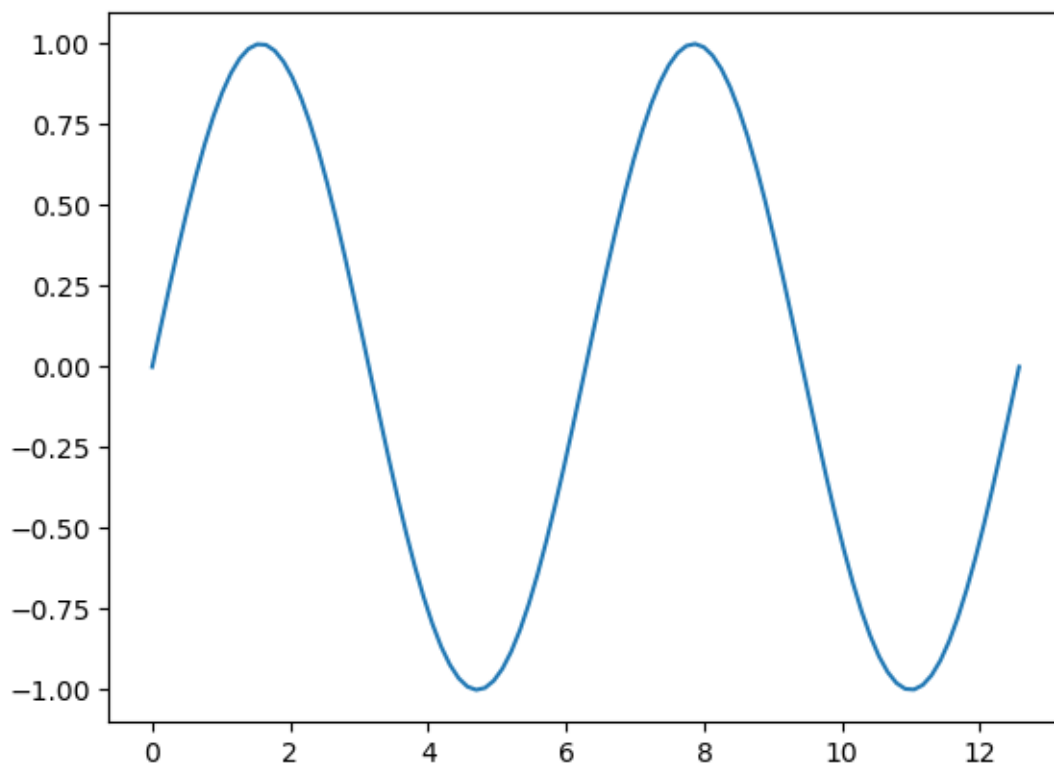
## 24.2 Matplotlib: Einfacher Plot

- [Dokumentation von plt.plot\(\)](#)

```
[27]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

# Sinuskurve
x = np.linspace(0.0, 4.0*np.pi, 100)
y = np.sin(x)

plt.plot(x, y) # zeichnet den graphen
plt.show()    # erzeugt den gesamt-plot
```



## 24.3 Matplotlib: Einfaches Säulendiagramm

- [Dokumentation plt.bar\(\)](#)

```
[47]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

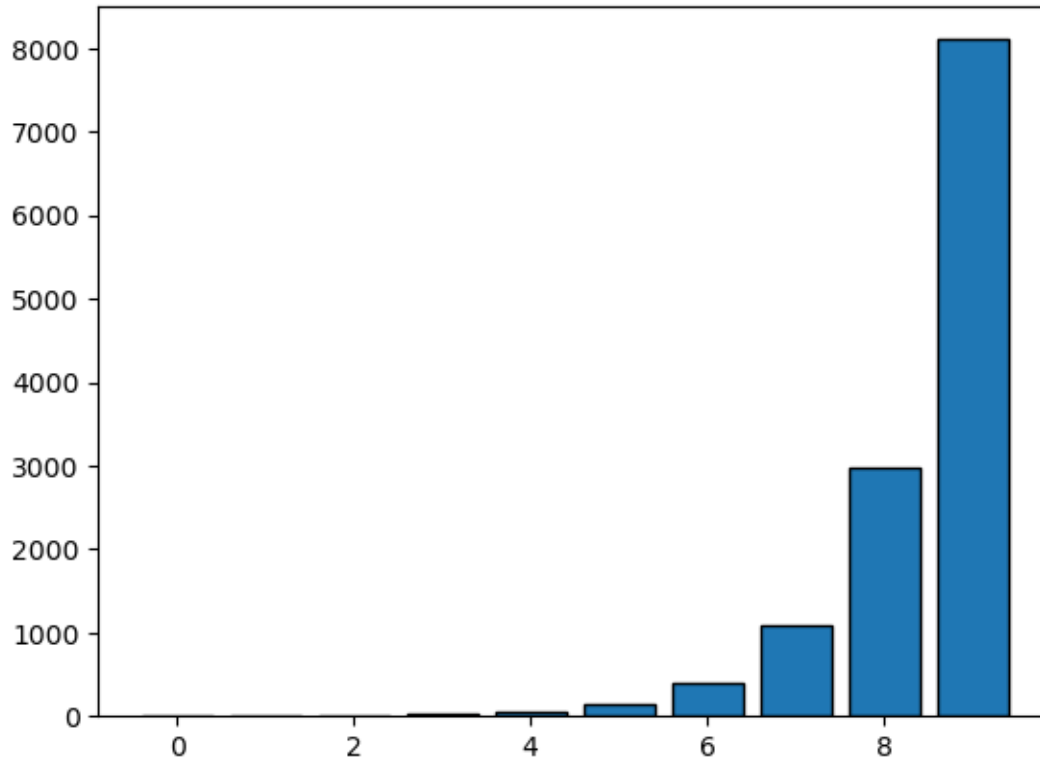
# Exponentialfunktion
x = np.arange(0, 10, 1)
```

```

y = np.exp(x)

plt.bar(x, y, edgecolor='k') # zeichnet die Säulen
plt.show()                  # erzeugt den gesamt-plot

```



## 24.4 Matplotlib: mehrere Plots mit Legende

```

[57]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

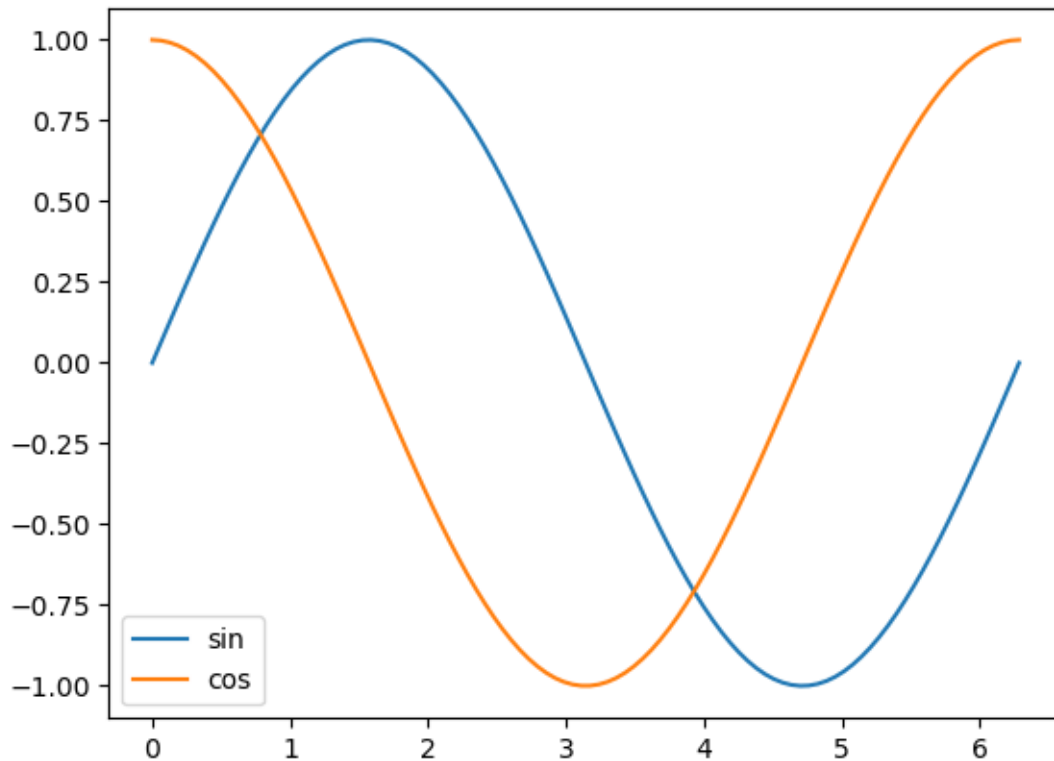
# Sinuskurve
x = np.linspace(0.0, 2.0*np.pi, 100)
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.plot(x, y_sin, label="sin") # zeichnet den graphen
plt.plot(x, y_cos, label="cos")

plt.legend() # erstellt die legende

plt.show()  # erzeugt den gesamt-plot

```



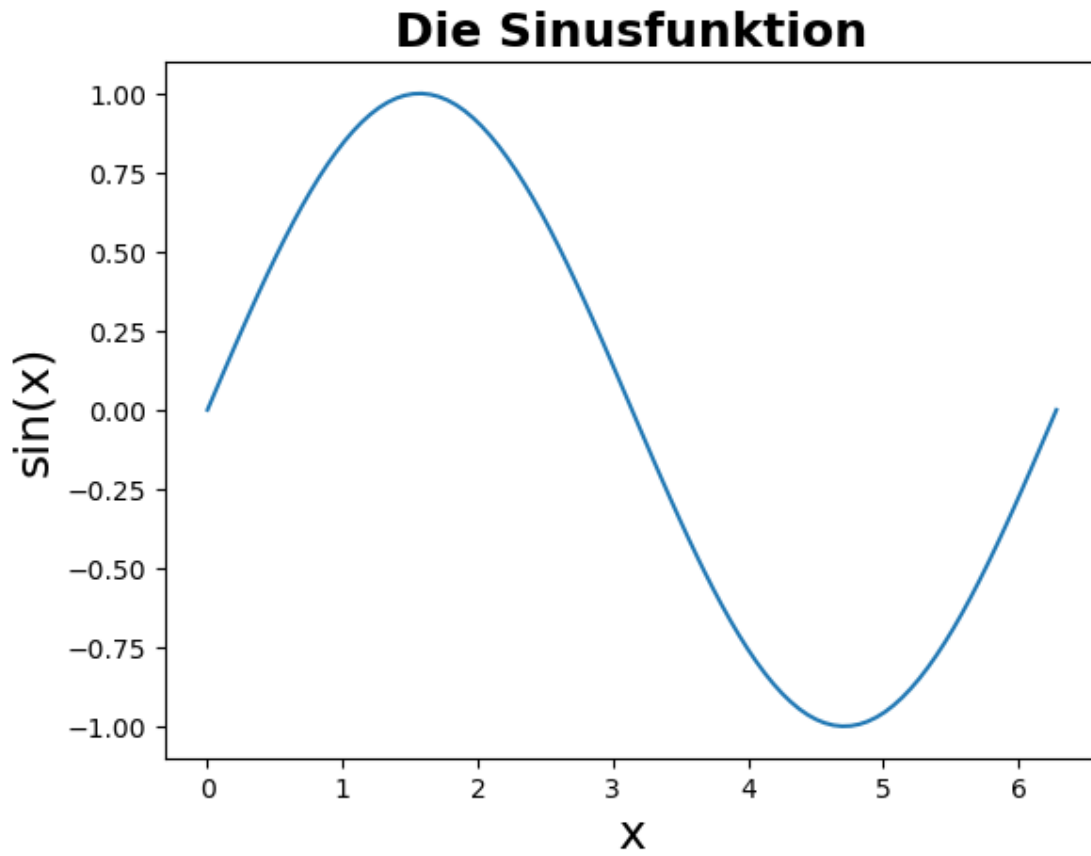
## 24.5 Matplotlib: Achsenbeschriftung

```
[29]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

# Sinuskurve
x = np.linspace(0.0, 2.0*np.pi, 100)
y_sin = np.sin(x)
plt.plot(x, y_sin) # zeichnet den graphen

""" erstellt die Beschriftung """
plt.xlabel("x", fontsize=18)
plt.ylabel("sin(x)", fontsize=18)
plt.title("Die Sinusfunktion", fontsize=18, fontweight="bold")

plt.show() # erzeugt den gesamt-plot
```



## 24.6 Matplotlib: Achsen Grenzen und Skalierung

```
[62]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

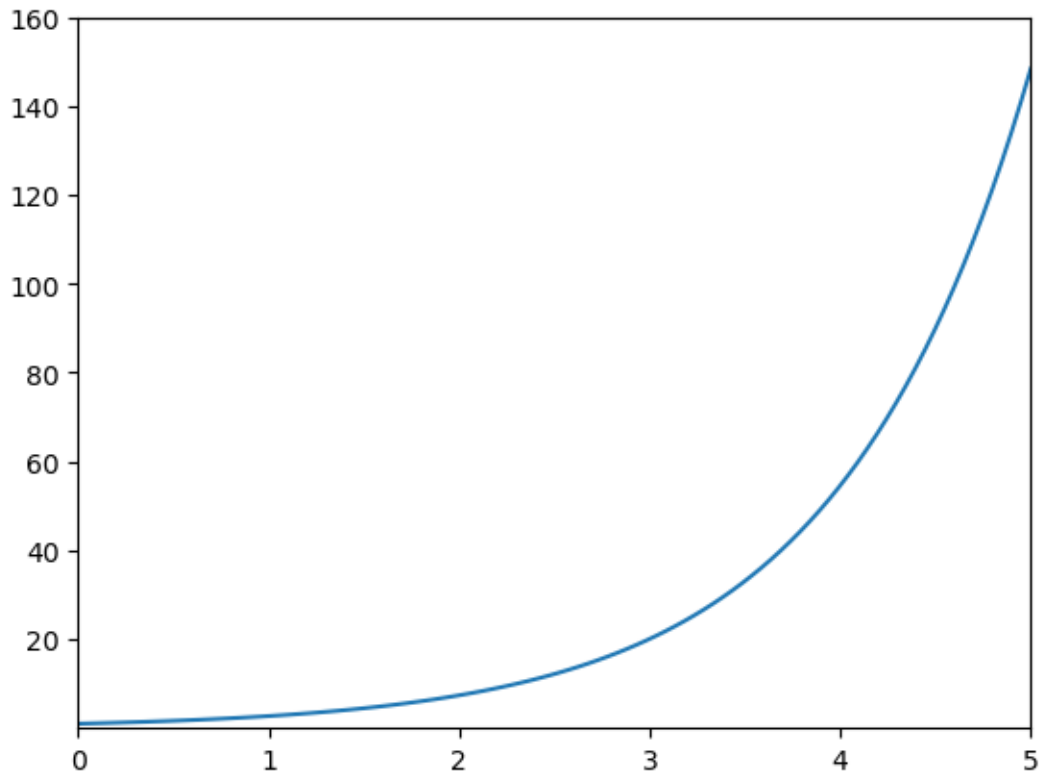
# Exponentialfunktion
x = np.linspace(0.0, 5, 100)
y = np.exp(x)

plt.plot(x, y) # zeichnet den graphen

""" gibt die Grenzen der Achse an """
plt.xlim(left=0.0, right=5)
plt.ylim(bottom=0.001, top=160)

""" Skalierung der Achse (logarithmisch oder normal)"""
#plt.yscale("log")

plt.show() # erzeugt den gesamt-plot
```



## 24.7 Matplotlib: Plot Einstellungen

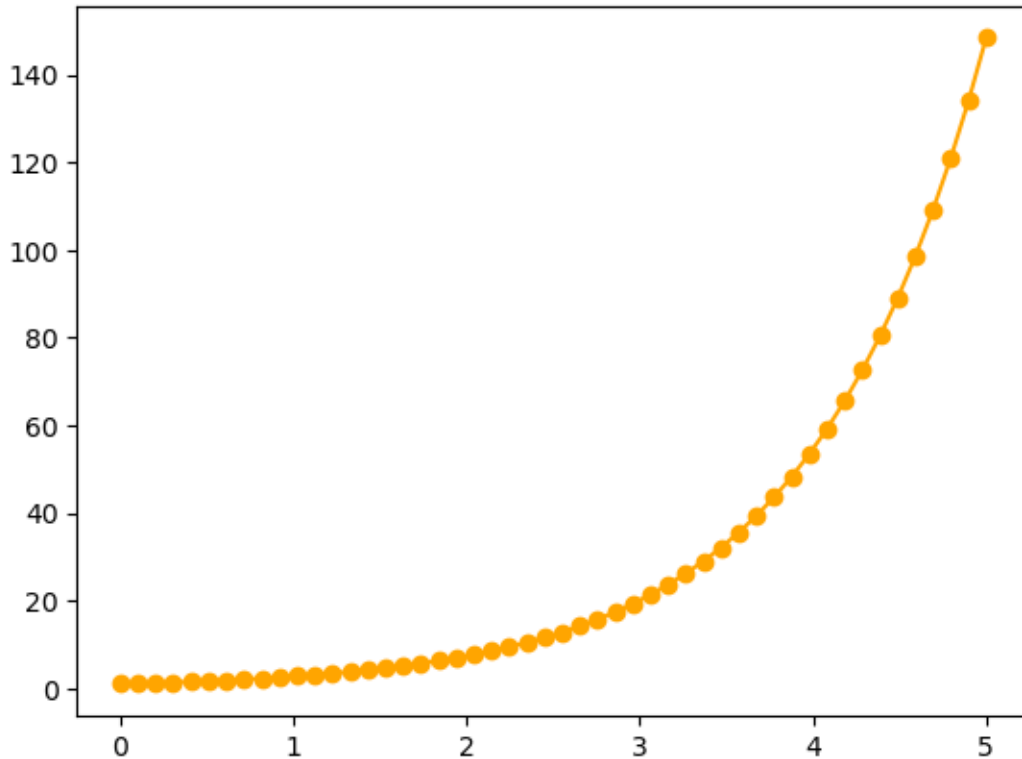
- für mehr einstellungen: **Matplotlib website besuchen**
- [Farben in Matplotlib](#)
- [Dokumentation von plt.plot\(\)](#)

```
[31]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

# Exponentialfunktion
x = np.linspace(0.0, 5, 50)
y = np.exp(x)

""" optionale Argumente für die Darstellung des Graphen"""
plt.plot(x, y, linestyle="-", color='orange', marker="o")

plt.show()      # erzeugt den gesamt-plot
```



## 24.8 Matplotlib: Übung

[Link zur Übung](#)

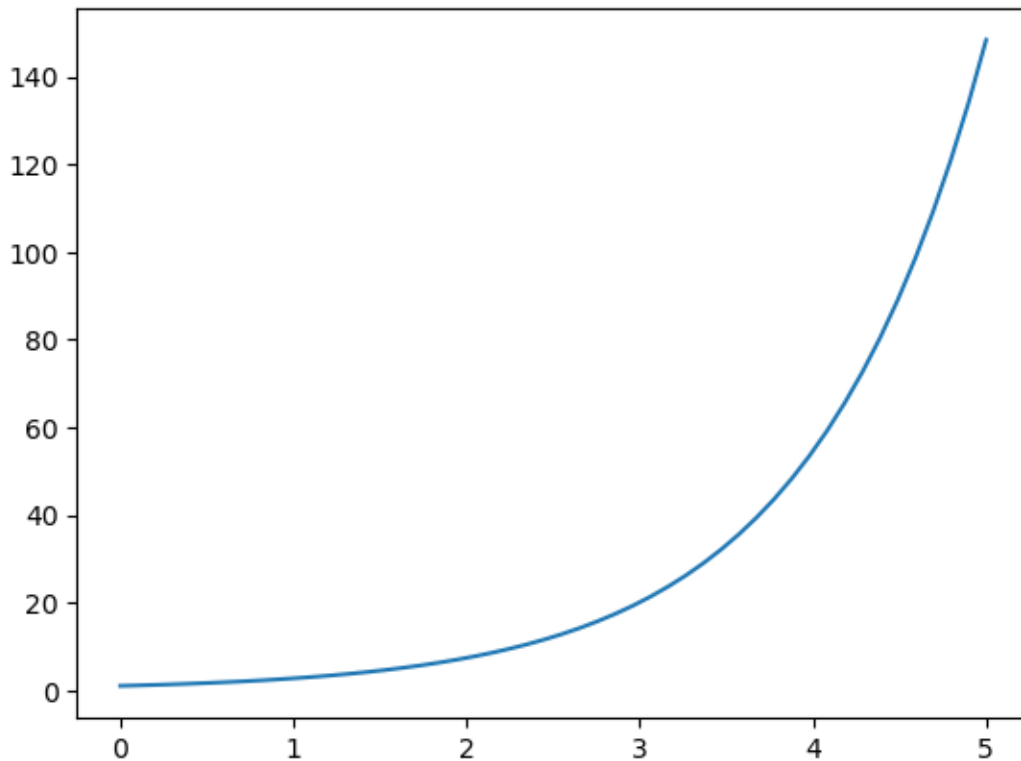
## 24.9 Matplotlib: Plot speichern

- **interaktiv**
  - “Speichern” Button drücken
  - rechts unten im Plot-Fenster
- **im Skript**
  - mit `plt.savefig("filename.ending")` speichern
  - "ending" kann "png", "jpg", "svg", "pdf" und vieles mehr sein
  - bei Pixelgraphiken (z.B. "png"): Auflösung mit Argument `dpi`

```
[32]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

# Exponentialfunktion
x = np.linspace(0.0, 5, 50)
y = np.exp(x)
plt.plot(x, y)

plt.savefig("exp_funktion.png", dpi=200)
```



## 24.10 Matplotlib: Größe der Abbildung

- Alle Einstellungen zur Abbildung mit der Funktion `plt.figure()`
- die Größe wird mit dem Keyword Argument `figsize=(with, height)` übergeben
- Maßeinheit ist Zoll

```
[33]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

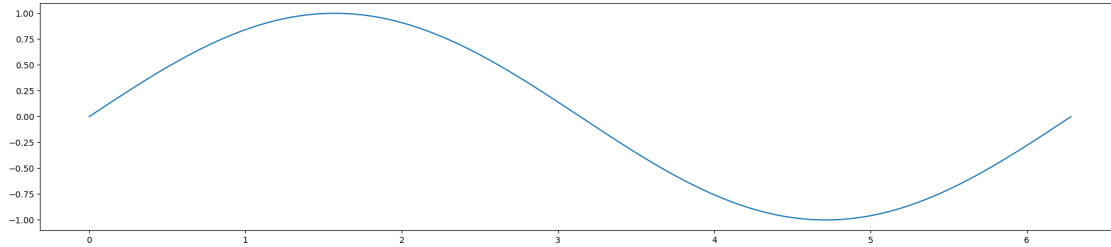
# Sinus und Cosinus
x = np.linspace(0.0, 2.0*np.pi, 200)
y_sin = np.sin(x)

plt.figure(figsize=(22.8, 4.8)) # ändert die Größe des Plots, default: 6.4x4.8
    ↳ Zoll

plt.plot(x, y_sin)

plt.show()
```





## 24.11 Matplotlib: Subplots

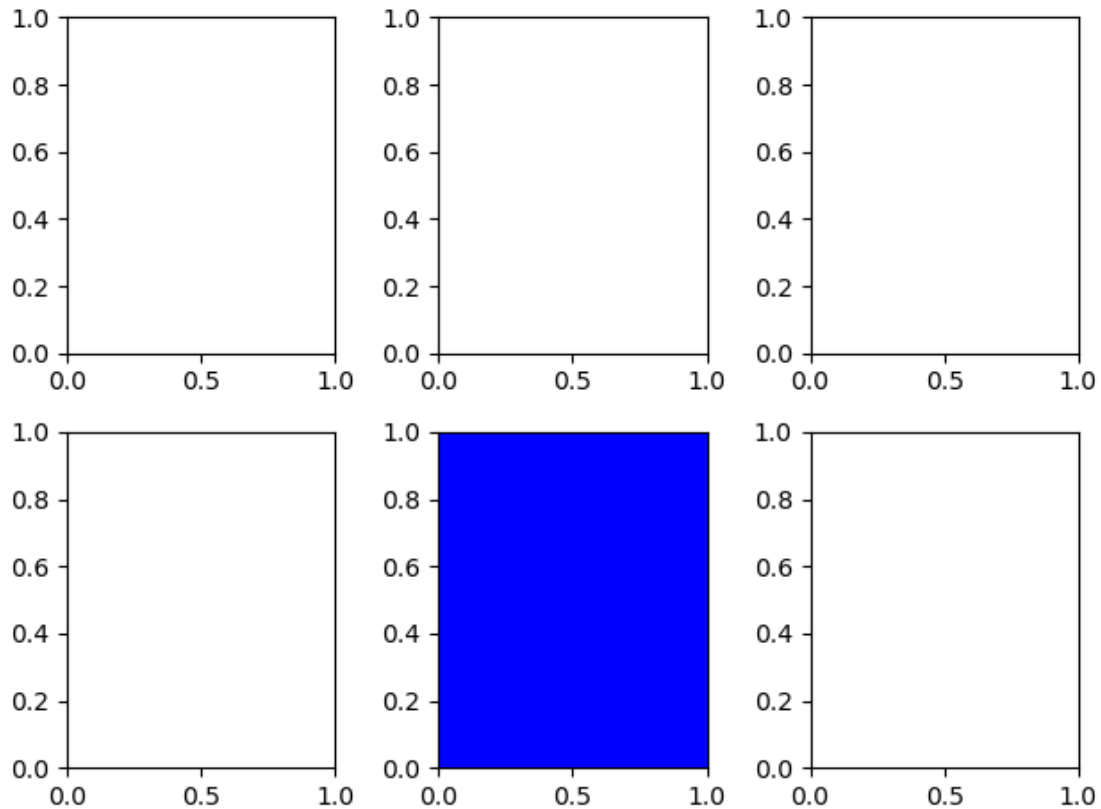
- mehrere Plots in einer Abbildung
- Gitter von Plots: (Anzahl Plots vertical) x (Anzahl Plots horizontal)
  - komplexere Anordnung auch möglich
- erzeugen von einem Subplot: `plt.subplot(id)`
- id ist dreistellige Zahl:
  - 1. Stelle: Anzahl Plots vertical
  - 2. Stelle: Anzahl Plots horizontal
  - 3. Stelle: Nummer des Plots
- Bsp. `plt.subplot(235)` [keepaspectratio]subplots.svg

```
[34]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

# Sinus und Cosinus
x = np.linspace(0.0, 2.0*np.pi, 200)
y_sin = np.sin(x)
y_cos = np.cos(x)

#plt.figure(figsize=(12.8, 4.8)) # ändert die Größe des Plots, default: 6.4x4.8
→ zoll

plt.subplot(231)      # plot Gitter 1x2, 1. Plot
plt.subplot(232)      # plot Gitter 1x2, 2. Plot
plt.subplot(233)
plt.subplot(234)
plt.subplot(235, fc='blue')
plt.subplot(236)
plt.tight_layout()    # verhindert Überlagerung, löscht "whitespace"
plt.savefig("subplots.svg")
```



## 24.12 Matplotlib: Subplots

```
[35]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

# Sinus und Cosinus
x = np.linspace(0.0, 2.0*np.pi, 200)
y_sin = np.sin(x)
y_cos = np.cos(x)

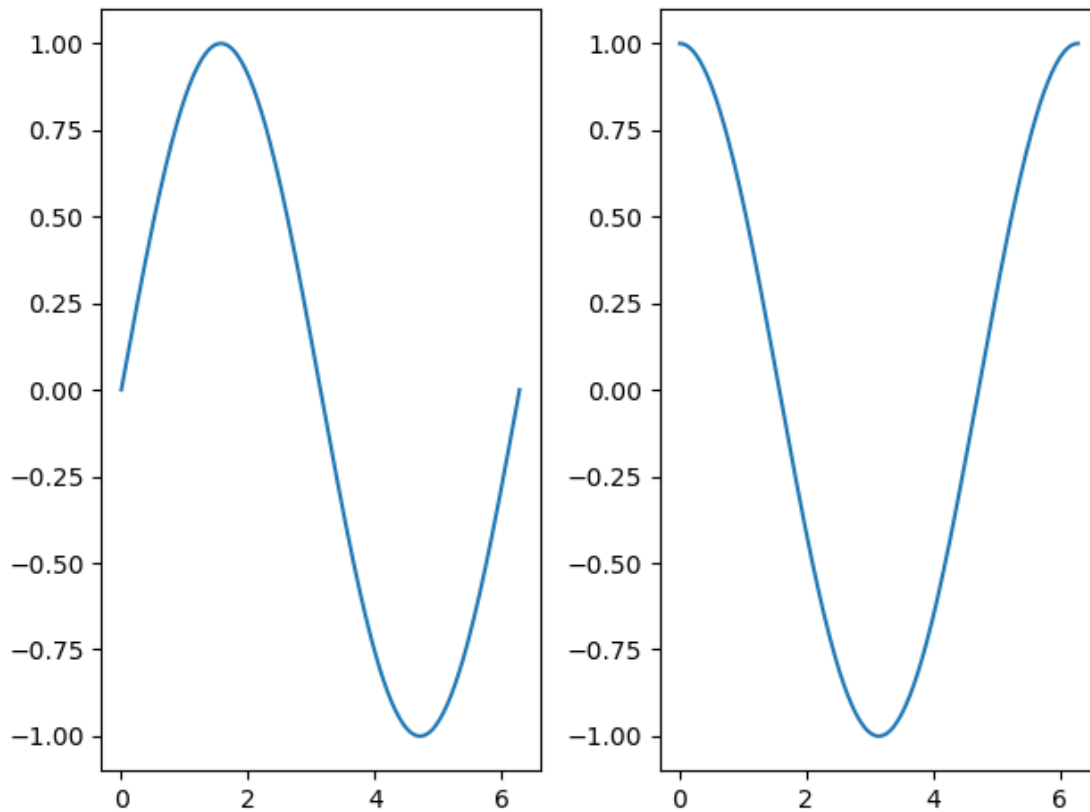
#plt.figure(figsize=(12.8, 4.8)) # ändert die Größe des Plots, default: 6.4x4.8
#  ↳ zoll

plt.subplot(121)      # plot Gitter 1x2, 1. Plot
plt.plot(x, y_sin)

plt.subplot(122)      # plot Gitter 1x2, 2. Plot
plt.plot(x, y_cos)

plt.tight_layout()    # verhindert Überlagerung, löscht "whitespace"
```

```
plt.show()
```



### 24.13 Matplotlib: Mehrere Abbildungen in einem Skript

- mehrere Abbildungen in einem Skript: Abbildungen müssen geschlossen werden
- nach jedem `plt.show()`: `plt.close("all")`
- verhindert das Überfüllen des Arbeitsspeichers (RAM)

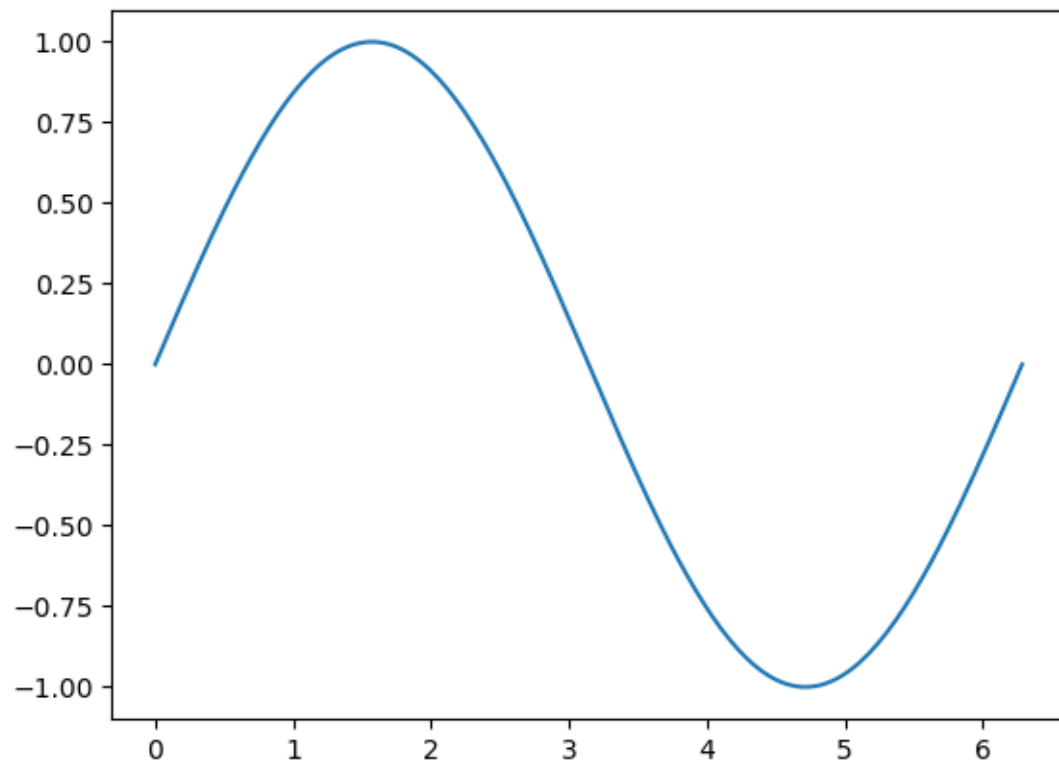
```
[36]: import numpy as np
import matplotlib.pyplot as plt # importieren der matplotlib bibliothek

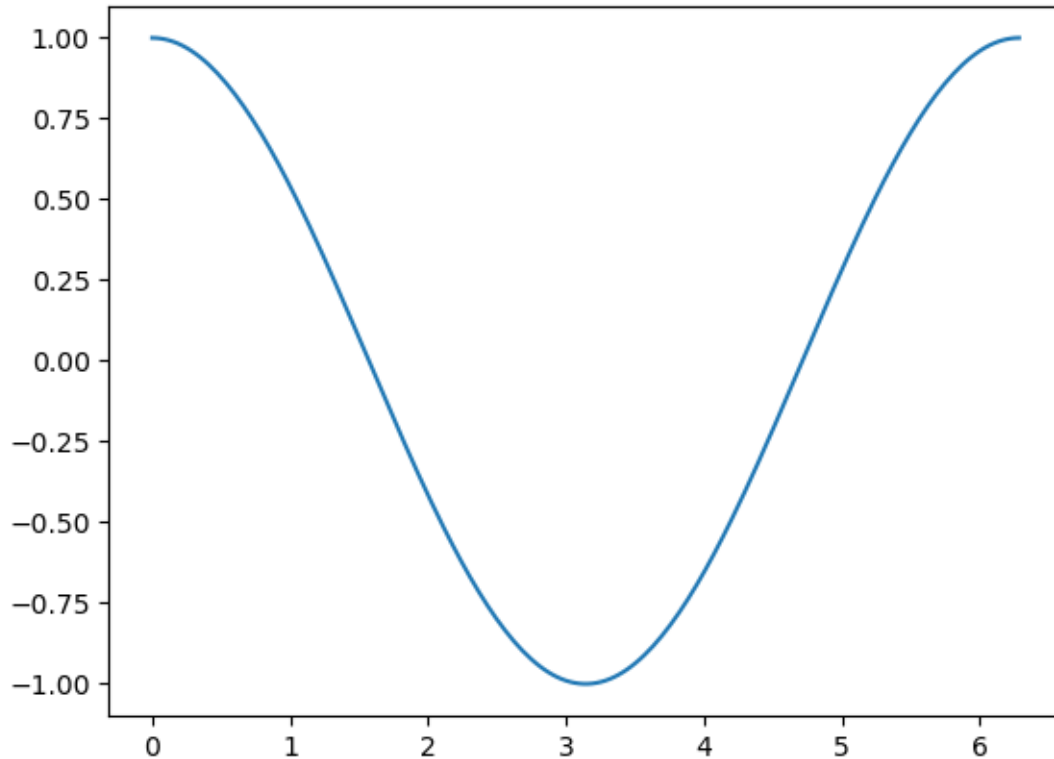
# Sinus und Cosinus
x = np.linspace(0.0, 2.0*np.pi, 200)
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.plot(x, y_sin)
plt.show()
plt.close('all')

plt.plot(x, y_cos)
```

```
plt.show()
```





## 25 Scipy

### 25.1 Scipy: Überblick

- Wissenschaftliche Bibliothek in Python, die auf NumPy aufbaut und erweiterte Funktionen für technische und wissenschaftliche Berechnungen bietet.
- **Untermodule:**
  - `scipy.constants`: Physikalische und mathematische Konstanten.
  - `scipy.fft`: Fourier-Transformationen.
  - `scipy.integrate`: Integrationstechniken (z.B. `quad`, `odeint`).
  - `scipy.interpolate`: Interpolation und Spline-Anpassung.
  - `scipy.linalg`: Lineare Algebra (erweitert NumPy's `linalg`).
  - `scipy.optimize`: Optimierung und Wurzelsuche (z.B. `minimize`, `curve_fit`).
  - `scipy.signal`: Signalverarbeitung (z.B. Filterung, Signaltransformation).
  - `scipy.sparse`: Arbeiten mit spärlichen Matrizen.
  - `scipy.spatial`: Berechnungen in der räumlichen Geometrie (z.B. Distanzen, Delaunay-Triangulation).
  - `scipy.stats`: Statistische Berechnungen (z.B. Wahrscheinlichkeitsverteilungen, Tests).
- **Verwendung:**
  - Hauptsächlich für numerische Berechnungen in Bereichen wie Physik, Chemie, Ingenieurwesen und Statistik.
  - Optimierung von Funktionen und Datenanalyse.

## 25.2 Scipy Beispiel: Lineare Regression

- mit der Funktion:

```
slope, intercept, r_value, p_value, std_err = scipy.stats.linregress(x, y)
```

- Rückgabewerte:
  - slope: Anstieg
  - intercept: Achsenabschnitt
  - r\_value: Pearson-Korrelationskoeffizient (Bestimmtheitsmaß:  $r\_value^2$ )

```
[37]: import numpy as np
import scipy as sc

# Beispiel-Daten generieren
x = np.linspace(0, 10, 50)
y = 2 * x + 1 + np.random.normal(0, 1, x.size) # y = 2x + 1 mit Rauschen

# Lineare Regression berechnen
slope, intercept, r_value, p_value, std_err = sc.stats.linregress(x, y)

print(f"Steigung: {slope:.2f}\nAchsenabschnitt: {intercept:.2f}\nR^2:␣
↪{r_value**2:.2f}")
```

Steigung: 1.99

Achsenabschnitt: 0.84

R^2: 0.96

## 25.3 Scipy: Lineare Regression Plotten

```
[38]: import numpy as np
import scipy as sc
import matplotlib.pyplot as plt

# Beispiel-Daten generieren
x = np.linspace(0, 10, 50)
y = 2 * x + 1 + np.random.normal(0, 1, x.size) # y = 2x + 1 mit Rauschen

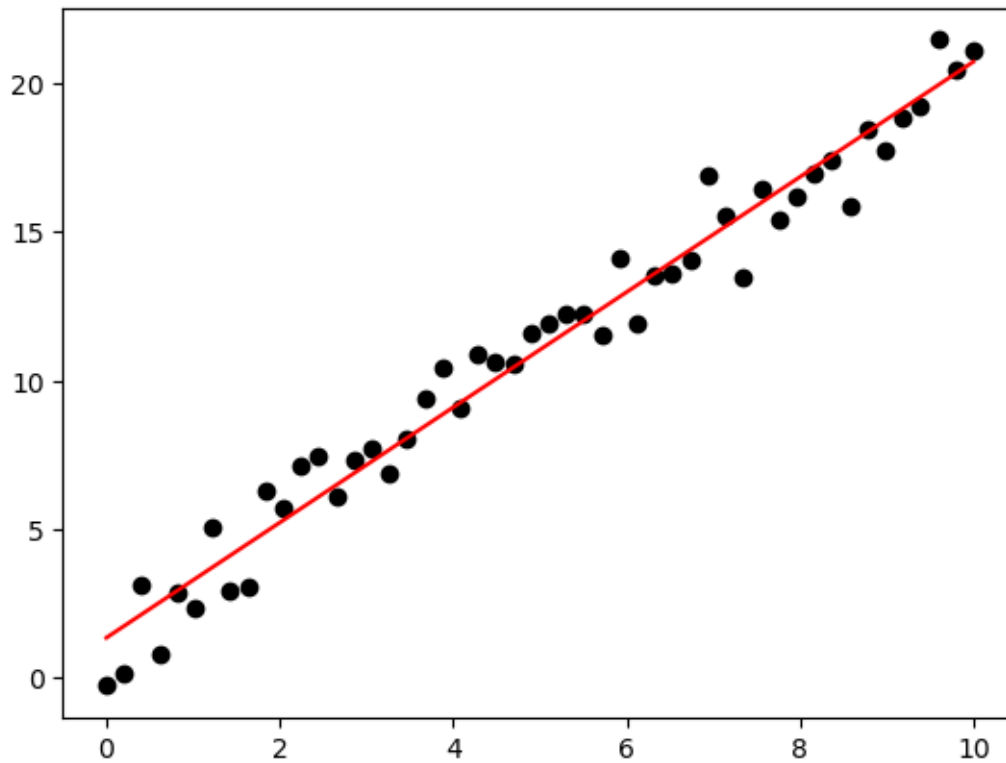
# Lineare Regression berechnen
slope, intercept, r_value, p_value, std_err = sc.stats.linregress(x, y)
print(f"Steigung: {slope:.2f}\nAchsenabschnitt: {intercept:.2f}\nR^2:␣
↪{r_value**2:.2f}")

plt.plot(x, y, color='black', linestyle='', marker='o')
plt.plot(x, slope * x + intercept, color='red')
plt.show()
```

Steigung: 1.94

Achsenabschnitt: 1.35

$R^2$ : 0.97



## 26 Zusammenfassung

- **Numpy**: Daten laden, Array-Operationen, mathematische Funktionen
- **Matplotlib**: Abbildungen erstellen
- **Scipy**: komplexere mathematische/physikalische Probleme
- **Übung macht den Meister**
  - manchmal Excel mit Python ersetzen