7_Bibliotheken_einbinden

October 26, 2021

1 Python-Bibliotheken für *DataScience* und sonstige Aufgaben

In diesem Notebook findest du einige Anmerkungen:

- Was sind Bibliotheken?
- Wie benutzt man Bibliotheken?
- Beispiele wichtiger Bibliotheken für die weiteren Notebooks und kurze Beschreibungen
  - Hier sind nur kurze Anmerkungen zu den Bibliotheks-Funktionen zu finden; doch zu allen hier benutzten Bibliotheken finden sich im Internet ausführlich Dokumentationen.

1.1 Einführung

Eine Programmierumgebung wie *Python* stellt neben den Basis-Datentypen und den Rechenoperationen (wie Addition, ganzzahlige Division etc.) eine Menge von Basis-Funktionen zur Verfügung. Man denke dabei z.B. an die `print`-Funktion, die jederzeit aufrufbar ist. Also:

- **Die Python-Standardbibliothek liefert viele Funktionen für gängige Programmieraufgaben.**

Doch daneben gibt es viele weitere Bibliotheken, in denen Funktionen bereitgestellt werden, die in einem Python-Programm nutzbar sind, so dass man als Programmierer nicht immer wieder *das Rad neu erfinden* muss.

Viele solcher Bibliotheken (in Python auch ***Module*** genannt) sind frei zugänglich, zum großen Teil bereits bei der Installation von Python eingerichtet oder können nachgeladen werden. Daneben gibt es Module von Drittanbietern.

Ebenfalls besteht die Möglichkeit, eigene Module zu entwickeln, die Funktionalitäten beinhalten, die wir selber entwickelt haben und dann in verschiedenen Python-Programmen nutzen können.

1.2 Eigene Bibliothek

Stell dir vor, du benötigst immer wieder gewisse mathematisch-interessante Funktionen, und möchtest dir eine Sammlung solcher Funktionen anlegen, die du bei Bedarf nutzen kannst, ohne den Code immer wieder neu schreiben zu müssen (oder per *copy-paste* zu nutzen).

Dann solltest du eine Python-Datei, z.B. unter dem Namen `meineBib.py` erzeugen, in der die Funktionen definiert sind.

Ein Beipiel:

```python
# meinBib.py

def fak (n):
    if n == 1:
        return 1
    else:
        return n * fak (n-1)

def kehrwert (x):
    return 1/x
```

Speichere diese Datei in deinem Arbeitsverzeichnis ab.

Jetzt kannst du auf die beiden dort definierten Funktionen zugreifen. Dazu kannst du wählen: - Alle Funktionen, die in diesem Modul definiert sind, werden zugänglich gemacht. - Dann werden die Funktionen mit dem sog. *qualifizierten Namen* aufgerufen:

```
    import meineBib
    print (meineBib.fak (10))
```

- Man kann dann den Bibliotheksnamen ggf. abkürzen:

```
    import meineBib as mb
    print (mb.fak (10))
```

- Nur einzelne Funktionen sind nutzbar.
  - Dann importiert man nur die benötigte(n) Funktion(en) und kann sie *unqualifiziert* nutzen
    ```
    from meineBib import fak
    print (fak (10))
    ```
  - Jedoch kann man dann eine Funktion mit gleichem Namen aus einer anderen Bibliothek nicht gleichzeitig nutzen!

1.3 Abkürzungen für Module: Schlüsselwort `as`

Mit dem Schlüsselwort`as` kann der Name einer zu importierenden Bibliothek umbenannt werden. Grund dafür könnte sein: - Der Name der Biobliothek wurde bereits für andere Zwecke benutzt (sehr unschöner Stil!!) - Eine andere Bibliothek hat denselben Namen (ebenfalls nicht sehr schön!) - Der Name der Bibliothek ist zu lang

[1]:
```
import meineBib as mb
print (mb.fak(10))
```

```
3628800
```

[2]:
```
from meineBib import fak
print (fak (10))
```

```
3628800
```

Möchte man alle Funktionen einer Bibliothek unqualifiziert importieren:

```
from meineBib import *
print (kehrwert (10))
```

[3]:
```
from meineBib import *
print (kehrwert (10))
```

```
0.1
```

Hinweis: Wo sucht das Python-System die importierten Module?

Es gibt eine Reihenfolge vorgegeben, nach der verfahren wird, wenn ein Modul importiert werden soll:

1. Zunächst wird der lokale Programmordner durchsucht. Wenn ein solches lokales Modul existiert, wird dieses eingebunden und keine weitere Suche durchgeführt.
   - Also kann man ein eigenes Modul erstellen, das einen Namen hat, wie ein ggf. intern vorhandenes gleichnamiges Modul.
2. Wenn kein lokales Modul mit dem angegebenen Namen gefunden wurde, wird ein global-vorhandenes Modul gesucht.
3. Wenn auch das nicht gefunden wurde, wird ein ModuleNotFoundError erzeugt:

1.4 Untermodule

Man kann auch mehrere Python-Bibliotheken aus inhaltlichen Gründen in einem Unterverzeichnis zusammenfassen. Schau dir dazu die folgenden Zeilen sowie die Bibliotheken dazu an:

[4]:
```
import myFunctions.meineBib1 as mb1

mb.fak (10)
```

[4]:
```
3628800
```

[5]:
```
import myFunctions as mf
mf.meineBib1.kehrwert (3)
```

[5]:
```
0.3333333333333333
```

[6]:
```
import myFunctions.meineBib2 as greeting
```

[7]:
```
greeting.gruss("Erna")
```

```
Hallo Erna
```

1.5 Fremde Bibliotheken

Sehr oft benutzt man Module, die aus anderen Quellen stammen.

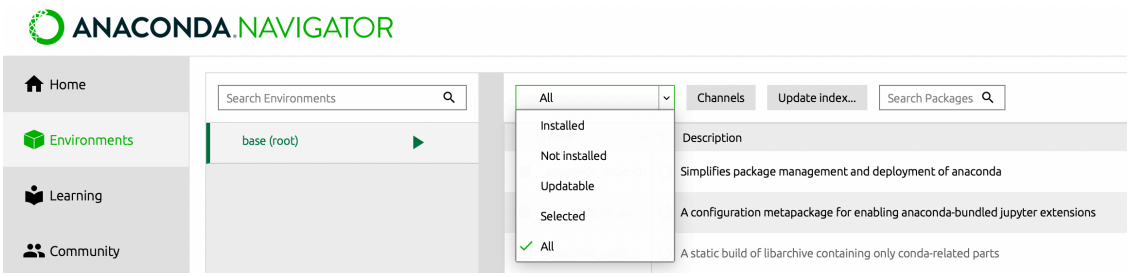
Wenn man die Anaconda-Installation benutzt, kann man auf eine Vielzahl solcher Module zurückgreifen. Eine Übersicht über

- bereits installierte
- mögliche weitere installierbare

Module kann man über die Anaconda-Oberfläche bekommen.

Dort kann man dann

- neue Module installieren
- neue Versionen von bereits installierten Modulen laden
- Module entfernen



Die Syntax für den Import eines solchen Moduls ist identisch zu dem Import eigener Bibliotheken.

Hinweis:

Man sollte regelmäßig die Versionen der benutzten Module überprüfen.

Alle externen, über Anaconda importierten Module haben eine Versionsnummer, die man in der Anaconda-Oberfläche erkennen kann. Es gibt auch die Möglichkeit, per Python-Anweisung diese Versionsnummer zu erfragen.

Am Beispiel der Bibliothek `numpy` (ein wichtige Bibliothek, die wir im folgenden Abschnitt genauer kennen lernen) sehen wir, wie man Fremdbibliotheken nutzt:

1.6 Die Bibliothek *numpy*

[12]:
```
import numpy as np
```

Die Versionsnummer anzeigen lassen:

[14]:
```
np.__version__
```

[14]:
```
'1.21.2'
```

Dieses Modul stellt Funktionen für die numerische Analyse von Daten zur Verfügung.

Insbesondere ermöglicht es den einfachen Umgang mit Arrays (Listen) und Matrizen. Es beinhaltet viele effizient implementierte Funktionen für numerische Berechnungen.

Der zentrale Datentyp in numpy ist `ndarray` (Abk. für *n-dimensionales Array*)

An einem einfachen Beispiel soll die Benutzung demonstriert werden.

Dokumentation kann gezeigt werden:

[15]:
```
np?
```

Der Wechselkurs Euro –> Dollar ist zur Zeit 1:1.21. Wir wollen eine Tabelle anlegen, aus der man für einige Euro-Beträge den zugehörigen Wert in Dollar ablesen kann.

[16]:
```
factor = 1.21

# es werden 10 Werte erzeugt im Intervall [100 ... 1000]; die Schritteweite␣
 ↪wird automatisch berechnet
#euros = np.linspace (start=100, stop = 1000, num = 10)

# alternativ:
# es werden Werte erzeugt im Intervall [100 ... 1000] mit Schrittweite 100

euros = np.arange(100, 1000, 100)

euros
```

[16]:
```
array([100, 200, 300, 400, 500, 600, 700, 800, 900])
```

Die Rechenoperationen `+`, `-`, `*` sowie viele mathematische Funktionen wie die Betragsfunktion `abs()`, die Wurzelfunktion `sqrt()` oder trigonometrische Funktionen wie `sin()`, `cos()` und `tan()`, können auf Numpy-Arrays angewendet werden. Dabei wird die Funktion auf jedes einzelne Element des Arrays angewendet. Als Ergebnis wird dann ein Array mit den entsprechenden Funktionswerten zurückgegeben.

[17]:
```
dollars = factor * euros # für unser Beispiel sinnvoll:

# erzeugt eine zwei-dimensionale Tabelle
table = np.array ([euros, dollars])

# Zeilen und Spalten transponiert, um besser lesen zu können
print (np.transpose (table))
```

```
[[ 100.  121.]
 [ 200.  242.]
 [ 300.  363.]
 [ 400.  484.]
 [ 500.  605.]
```

```
 [ 600.  726.]
 [ 700.  847.]
 [ 800.  968.]
 [ 900. 1089.]]
```

Man kann mit Hilfe von numpy auch (Pseudo-)Zufallszahlen erzeugen. Das geschieht mit der Funktion `random()` aus dem Untermodul `random`. Diese Funktion erzeugt die im Parameter angegebene Anzahl von zufälligen float-Werten im halboffenen Intervall [0, 1)

[18]:
```
zufListe = np.random.random(10)
print (zufListe)
```

```
[0.55748223 0.16808962 0.75615917 0.8970714  0.374212   0.5436296
 0.91338297 0.44913883 0.67149013 0.05703441]
```

1.7 Die Bibliothek *matplotlib* mit der Teilbibliothek *pyplot*

Diese Bibliothek gestattet es, Digramme zu erzeugen. Das kann z.B. auf der Grundlage von Python-Listen erfolgen:

[19]:
```
import matplotlib
```

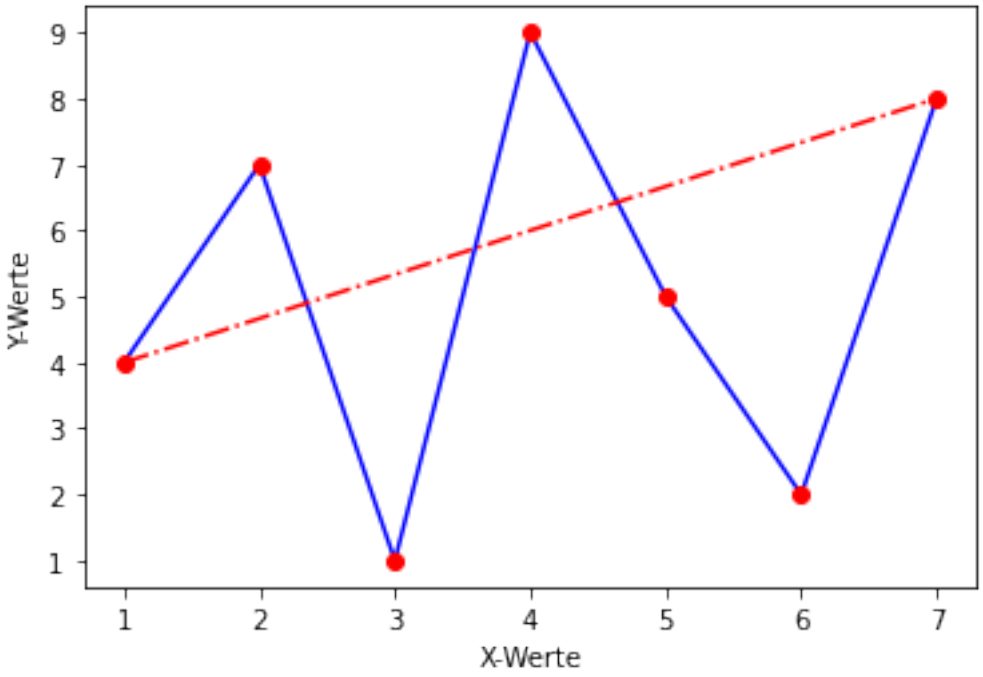
[20]:
```
matplotlib.__version__
```

[20]:
```
'3.4.3'
```

[21]:
```
import matplotlib.pyplot as plt
```

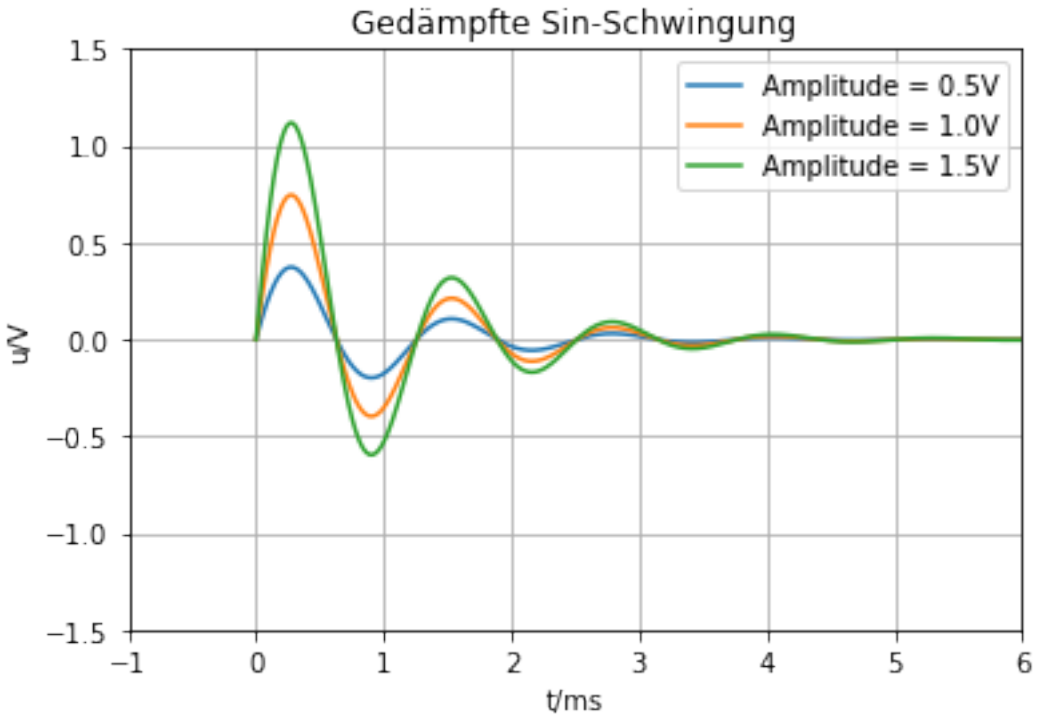
[22]:
```
plt?
```

[23]:
```
Y = [4, 7, 1, 9, 5, 2, 8]
X = [1, 2, 3, 4, 5, 6, 7]
plt.plot(X, Y,color='blue')
#plt.scatter(X, Y, color='red')
plt.plot (X, Y, 'ro')   # Alternativ
# Abkürzungen der Symbole sind z.B.
# ".", o", "+", "x" für die Punktsymbole sowie
# "r", "b", "m", "g" für Farben
plt.xlabel("X-Werte")
plt.ylabel("Y-Werte")

x1 = X[0]
x2 = X[-1]
y1 = Y[0]
y2 = Y[-1]
plt.plot((x1, x2), (y1, y2), 'r-.')
# bei der Linienart kann man z.B. '-', '-.' oder '--' nutzen

plt.show()
```



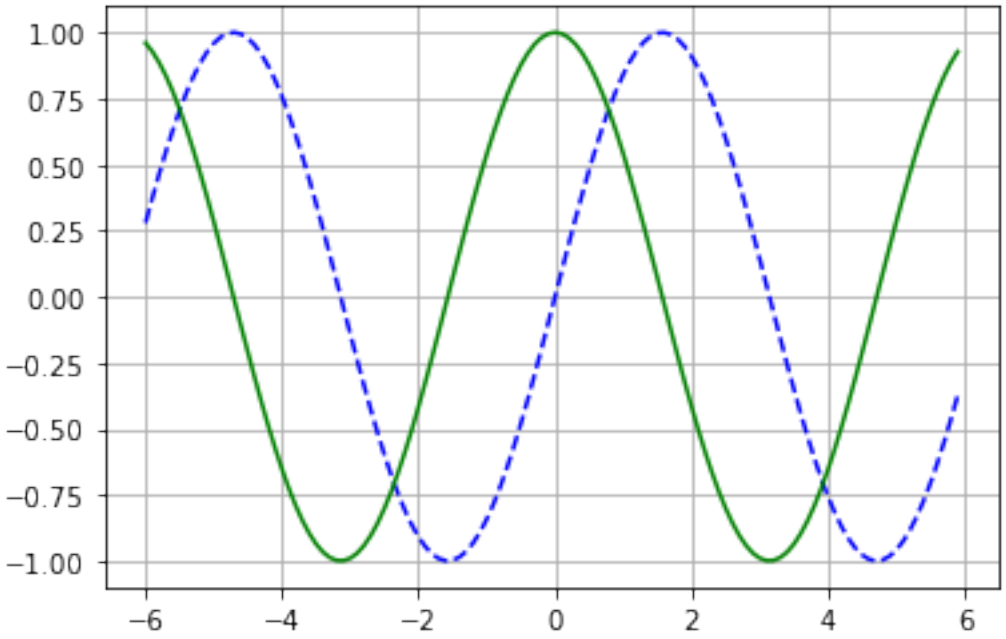
[24]:
```
from numpy import sin, exp, linspace, arange, pi
x=linspace(0.0, 2*pi, 1000)
for a in arange(0.5,2, 0.5):
    y = a * sin(5*x)*exp(-x)
    plt.plot(x, y, label = "Amplitude = " + str(a)+ "V" )
plt.grid(True)
plt.xlim(-1.0, 6.0)
plt.ylim(-1.5, 1.5)
plt.xlabel("t/ms")
plt.ylabel("u/V")
plt.title ("Gedämpfte Sin-Schwingung")
plt.legend()
plt.show
```

[24]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

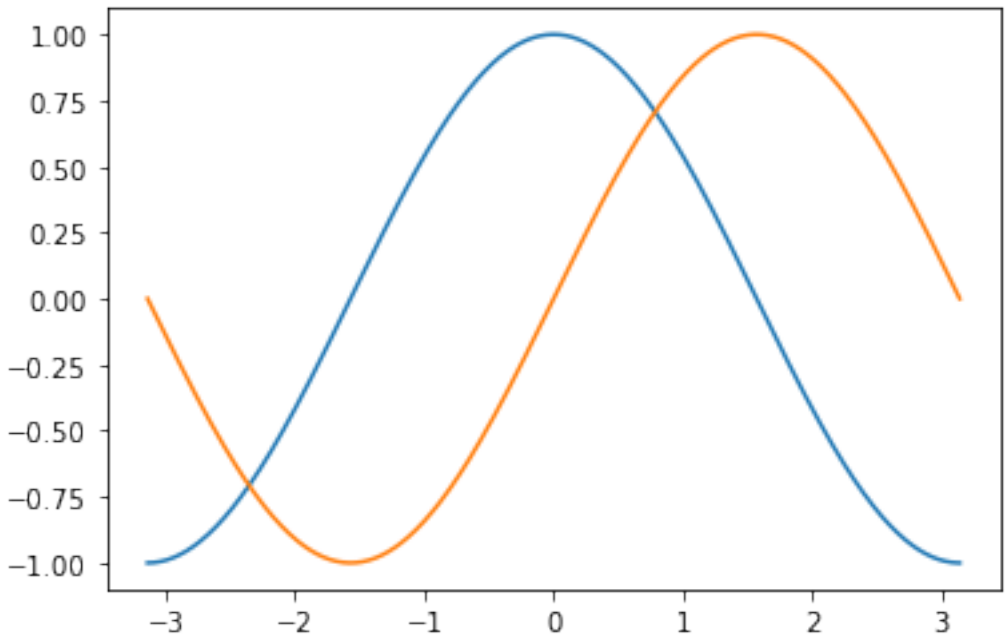


1.8 Ein weiteres Beispiel

[25]:
```
# Wertebereich für x-Achse festlegen:
x = np.arange (-6,6,0.1)

x2 = np.sin(x)
x3 = np.cos(x)

# Einzelne Diagramm-Linien plotten:
#plt.plot(x, x, 'r--')
plt.plot(x, x2,'b--')
plt.plot(x, x3,'g-')

# Diagramm-Gitter einblenden:
plt.grid(True)

# Diagramm ausgeben:
plt.show()
```

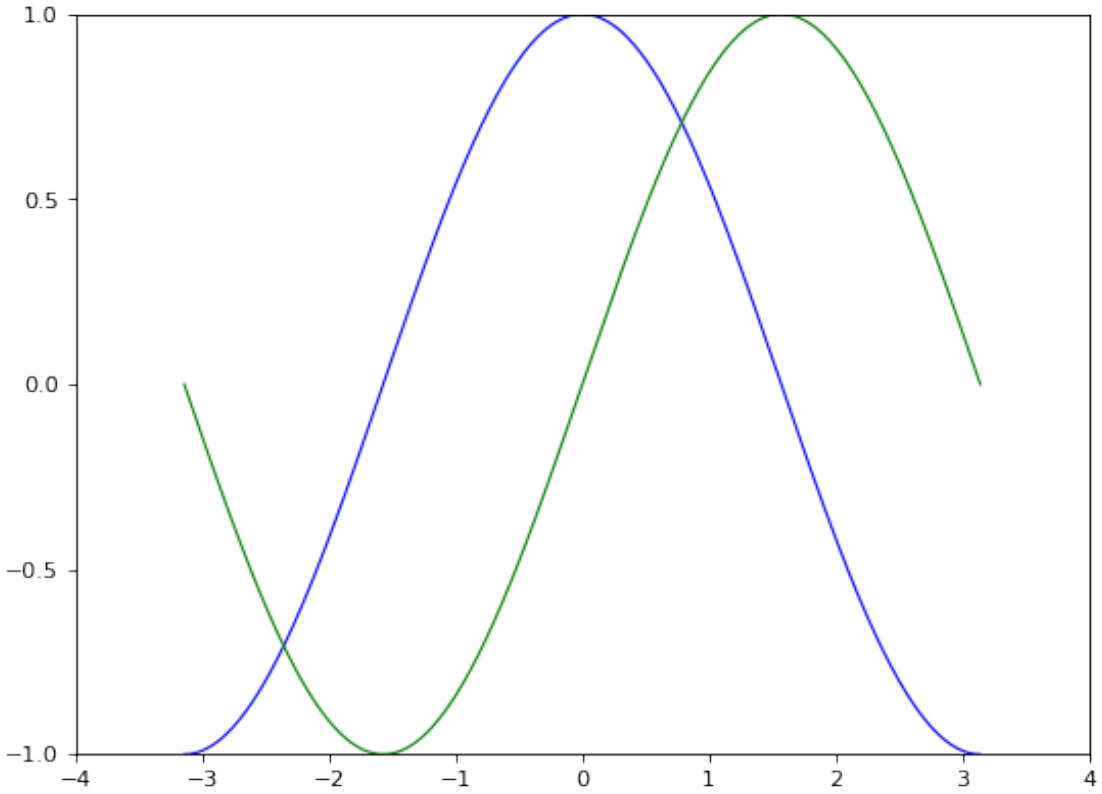


[26]:
```
# Werte-Listen für eine Sinus- und Cosinus-Funktion erstellen:
x = np.linspace(-np.pi, np.pi, 500, endpoint=True)
cos_x = np.cos(x)
sin_x = np.sin(x)

# Diagramm-Linien plotten:
plt.plot(x, cos_x)
plt.plot(x, sin_x)

# Diagramm anzeigen:
plt.show()
```



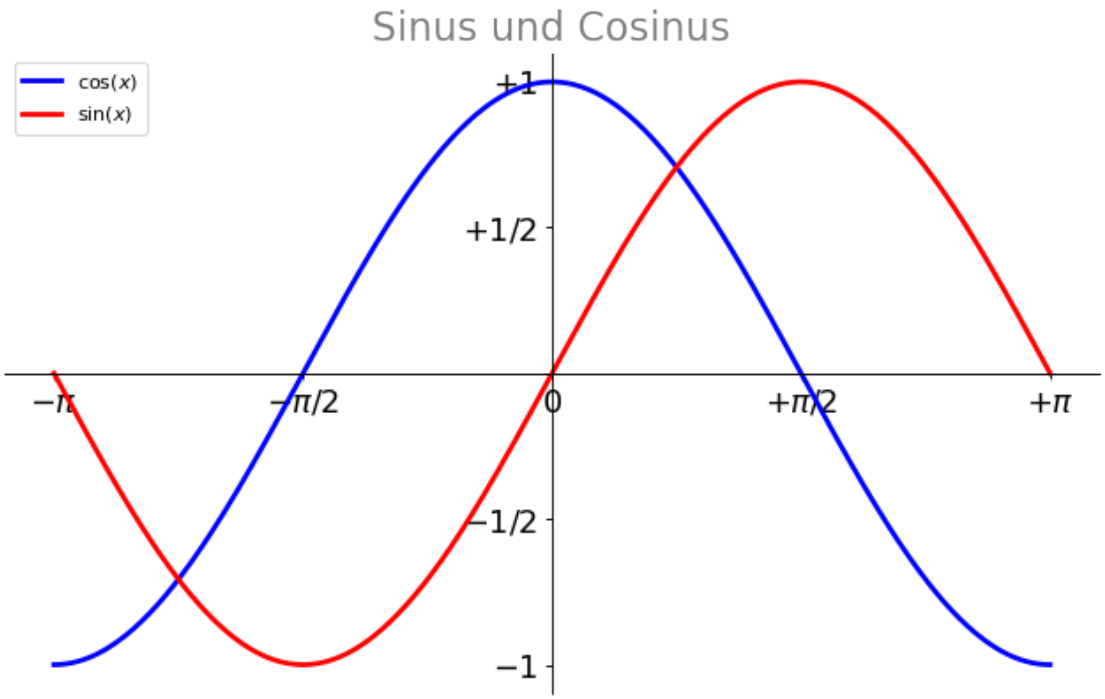
[27]:
```
# Werte-Listen für eine Sinus- und Cosinus-Funktion erstellen:
x = np.linspace(-np.pi, np.pi, 500, endpoint=True)
cos_x = np.cos(x)
sin_x = np.sin(x)

# Eine neues Matplot-Figure-Objekt mit 8x6 Zoll und
# einer Auflösung von 100 dpi erstellen:
plt.figure(figsize=(8, 6), dpi=80)

# In diese Abbildung ein 1x1 großes Diagramm-Gitter erstellen;
# Als aktuelles Diagramm wird das erste dieses Gitters ausgewählt:
plt.subplot(111)

# Cosinus-Funktion mit blauer Farbe, durchgehender Linie und 1 Pixel
# Linienbreite plotten:
plt.plot(x, cos_x, color="blue", linewidth=1.0, linestyle="-")

# Sinus-Funktion mit grüner Farbe, durchgehender Linie und 1 Pixel
# Linienbreite plotten:
plt.plot(x, sin_x, color="green", linewidth=1.0, linestyle="-")

# Grenzen für die x-Achse festlegen:
plt.xlim(-4.0, 4.0)

# Grenzen für die y-Achse festlegen:
plt.ylim(-1.0, 1.0)
```

```
# "Ticks" (Bezugspunkte) für x-Achse festlegen:
plt.xticks(np.linspace(-4, 4, 9,endpoint=True))

# "Ticks" (Bezugspunkte) für y-Achse festlegen:
plt.yticks(np.linspace(-1, 1, 5,endpoint=True))

# Diagramm anzeigen:
plt.show()
```



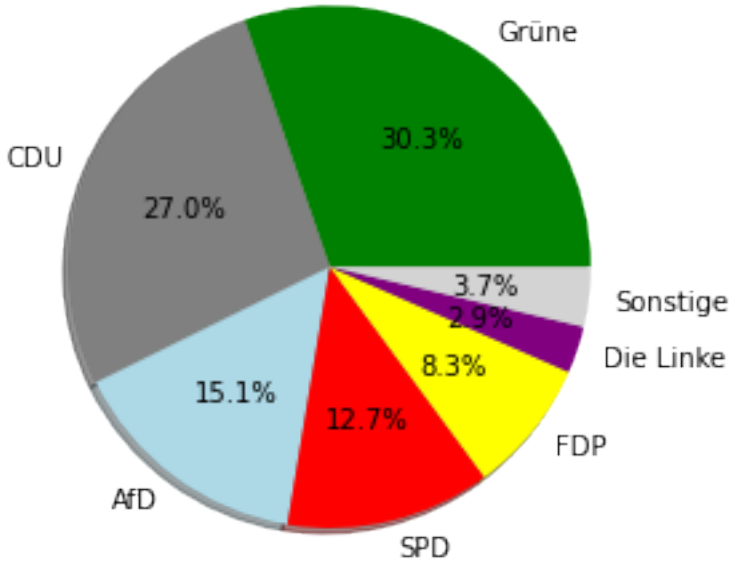
[28]:
```
# Größe des Plots anpassen:
plt.figure(figsize=(10,6), dpi=80)
# Farbe und Dicke der Diagrammlinien anpassen:
# Plots mit einem Label versehen:
plt.plot(x, cos_x, color="blue", linewidth=2.5, linestyle="-",␣
 ↪label=r'$\cos(x)$')
plt.plot(x, sin_x, color="red",  linewidth=2.5, linestyle="-",␣
 ↪label=r'$\sin(x)$')
# Wertebereiche der Achsen anpassen:
plt.xlim(x.min()*1.1, x.max()*1.1)
plt.ylim(cos_x.min()*1.1, cos_x.max()*1.1)
```

```
# Auf der x-Achse fünf Bezugspunkte (als Vielfache von pi) festlegen
# und mittels LaTeX-Symbolen beschriften:
plt.xticks( [-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
            [ r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$']
    )

# Auch Ticks für die y-Achse anpassen:
plt.yticks( [-1.0, -0.5, 0, 0.5, 1],
            [ r'$-1$', r'$-1/2$', r'', r'$+1/2$', r'$+1$']
    )

# Das Achsen-Objekt des Diagramms in einer Variablen ablegen:
ax = plt.gca()

# Die obere und rechte Achse unsichtbar machen:
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

# Die linke Diagrammachse auf den Bezugspunkt '0' der x-Achse legen:
ax.spines['left'].set_position(('data',0))

# Die untere Diagrammachse auf den Bezugspunkt '0' der y-Achse legen:
ax.spines['bottom'].set_position(('data',0))

# Ausrichtung der Achsen-Beschriftung festlegen:
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')

# Achse-Beschriftungen durch weiß-transparenten Hintergrund hervorheben:
for label in ax.get_xticklabels() + ax.get_yticklabels():
    label.set_fontsize(16)
    label.set_bbox(dict(facecolor='white', edgecolor='None', alpha=0.65 ))

# Titel hinzufügen:
plt.title('Sinus und Cosinus', fontsize=20, color='gray')

# Legende einblenden:
plt.legend(loc='upper left', frameon=True)


plt.show ()
```

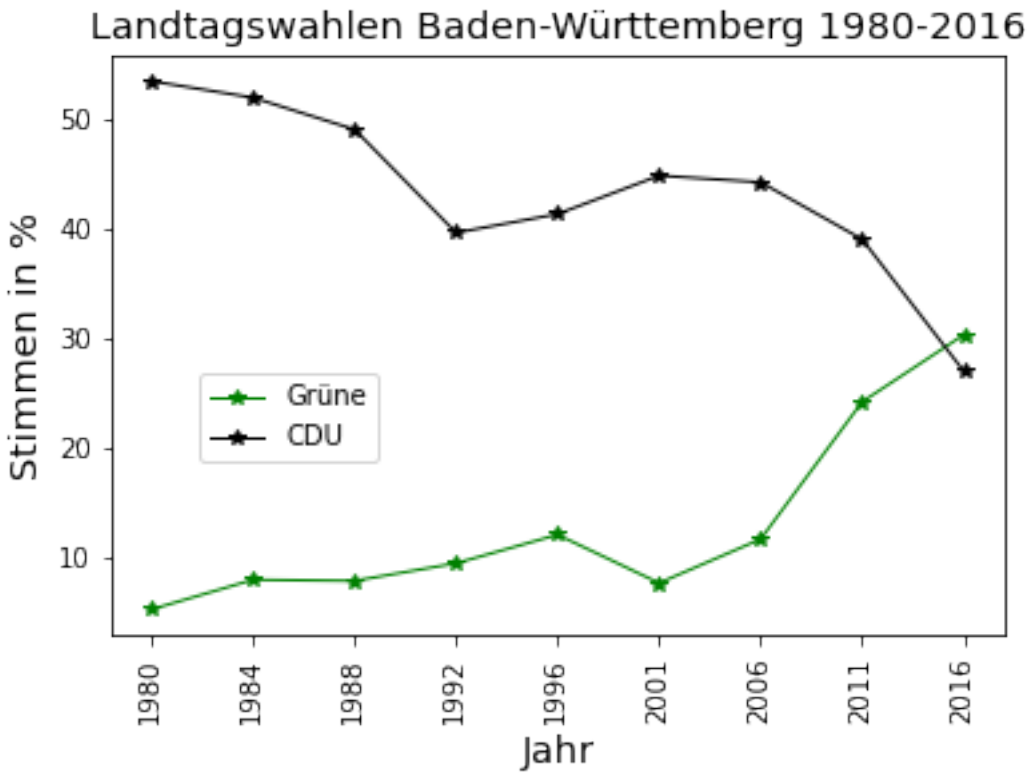


1.9 Weitere graphisch Darstellungen

[29]:
```
# matplotlib style
plt.style.use('seaborn-bright')

# data to plot
labels = 'Grüne', 'CDU', 'AfD', 'SPD', 'FDP', 'Die Linke', 'Sonstige'
sizes = [30.3, 27, 15.1, 12.7, 8.3, 2.9, 3.7]
colors = ['green', 'gray', 'lightblue', 'red', 'yellow', 'purple', 'lightgray']

# plot
plt.pie(sizes,              # data
        labels=labels,      # slice labels
        colors=colors,      # array of clors
        autopct='%1.1f%%',  # print the values inside the wedges
        shadow=True,        # enable shadow
        startangle=0)     # startin angle

plt.axis('equal')

plt.show()
```



[30]:
```
gruene = [5.3, 8.0, 7.9, 9.5, 12.1, 7.7, 11.7, 24.2, 30.3]
cdu = [53.4, 51.9, 49.0, 39.6, 41.3, 44.8, 44.2, 39.0, 27.0]

fig, ax = plt.subplots()
years = [1980, 1984, 1988, 1992, 1996, 2001, 2006, 2011, 2016]

plt.title("Landtagswahlen Baden-Württemberg 1980-2016", size="x-large")
plt.ylabel("Stimmen in %", size="x-large")
plt.xlabel("Jahr", size="x-large")

plt.plot(gruene, "*-", markersize=6, linewidth=1, color='green', label="Grüne")
plt.plot(cdu, "*-", markersize=6, linewidth=1, color='black', label="CDU")

plt.legend(loc=(0.1, 0.3))

ax.set_xticks(range(len(years)))
ax.set_xticklabels(years, rotation='vertical')

plt.show()
```



1.10 Die Bibliothek *pandas*

In einer Datenerhebung werden viele Datenpaare gesammelt, die in der Regel in einer ***csv***-Datei (z.B. unter dem Namen `abc.csv` in einem Unterordner `daten` des Arbeitsverzeichnisses) abgelegt sind.

`pandas` ist eine Programmbibliothek für die Programmiersprache Python, die Hilfsmittel für die Verwaltung von Daten und deren Analyse anbietet. Insbesondere enthält sie Datenstrukturen und Operatoren für den Zugriff auf numerische Tabellen und Zeitreihen.

(Wikipedia)

[31]:
```
import pandas as pd
```

[32]:
```
pd.__version__
```

[32]:
```
'1.3.3'
```

[33]:
```
df = pd.read_csv("daten/Auto.csv",sep=";",decimal=",",␣
 ↪index_col=["Hersteller","Modell"])
df
```

[33]:

Hersteller	Modell	Kilometerstand	Baujahr	PS	Zustand	Kraftstoff	Verbrauch	Preis
BMW	520	40000.0	2017.0	190	Gebraucht	Diesel	4.5	46900
Volvo	XC60	42656.0	2017.0	190	Gebraucht	Diesel	4.7	32930
	XC90	111183.0	2015.0	224	Gebraucht	Diesel	5.8	40930
Audi	Q3	42922.0	2016.0	184	Gebraucht	Diesel	5.2	30990
Seat	Ibiza	138000.0	2008.0	69	Gebraucht	Benzin	5.9	3200
Volkswagen	Tiguan	13189.0	2018.0	179	Gebraucht	Benzin	7.6	39930
	Touran	45000.0	2014.0	140	Gebraucht	Benzin	7.2	17000
	Passat	100.0	2018.0	67	Gebraucht	Benzin	4.7	10240
Audi	A3	345000.0	2006.0	170	Gebraucht	Diesel	5.7	3550
Volvo	V50	260000.0	2008.0	109	Gebraucht	Diesel	5.0	3200
Renault	Scenic	2000.0	2018.0	140	Neu	Benzin	5.5	24990
	Clio	2000.0	2018.0	90	Neu	Benzin	5.0	14990
Kia	Rio	4.0	NaN	84	Neu	Benzin	5.5	14250
Peugeot	307	52750.0	2004.0	109	Gebraucht	Benzin	NaN	2790
BMW	218	42346.0	2015.0	150	Gebraucht	Diesel	4.5	21950
Volkswagen	Caddy	156600.0	2007.0	105	Gebraucht	Diesel	6.9	2890
Fiat	500	6917.0	2013.0	69	Gebraucht	Benzin	5.1	9450
Mazda	6	NaN	NaN	194	Neu	Benzin	6.8	34190
	MX-5	NaN	NaN	132	Neu	Benzin	6.3	23190
Volkswagen	T6	26537.0	2017.0	204	Jahreswagen	Diesel	7.6	60990

[34]:
```
df.sort_index()
```

[34]:

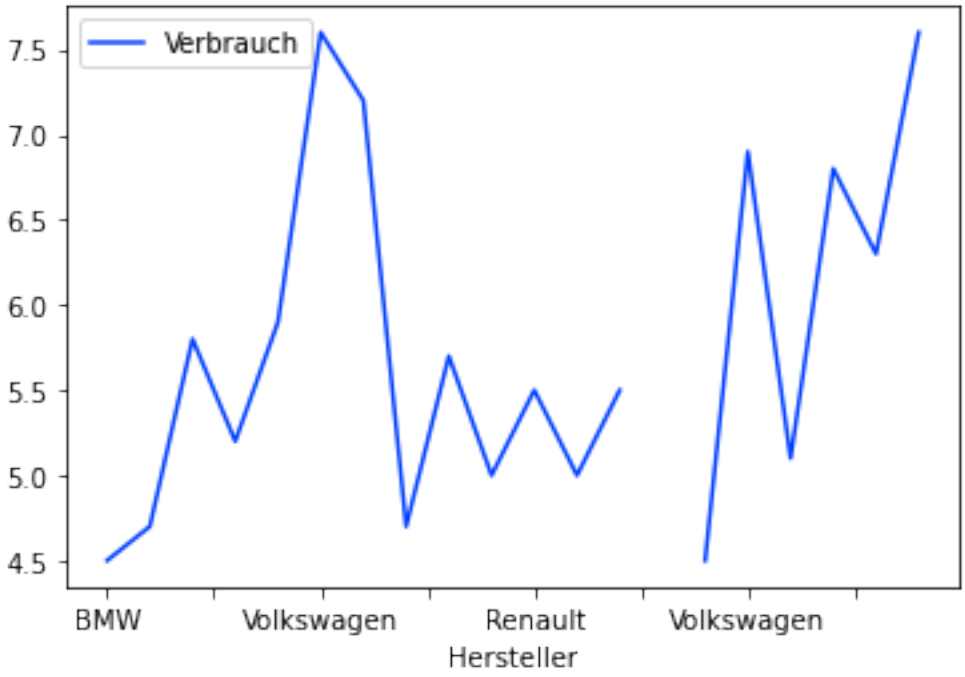
Hersteller	Modell	Kilometerstand	Baujahr	PS	Zustand	Kraftstoff	Verbrauch	Preis
Audi	A3	345000.0	2006.0	170	Gebraucht	Diesel	5.7	3550
	Q3	42922.0	2016.0	184	Gebraucht	Diesel	5.2	30990
BMW	218	42346.0	2015.0	150	Gebraucht	Diesel	4.5	21950
	520	40000.0	2017.0	190	Gebraucht	Diesel	4.5	46900
Fiat	500	6917.0	2013.0	69	Gebraucht	Benzin	5.1	9450
Kia	Rio	4.0	NaN	84	Neu	Benzin	5.5	14250
Mazda	6	NaN	NaN	194	Neu	Benzin	6.8	34190
	MX-5	NaN	NaN	132	Neu	Benzin	6.3	23190
Peugeot	307	52750.0	2004.0	109	Gebraucht	Benzin	NaN	2790
Renault	Clio	2000.0	2018.0	90	Neu	Benzin	5.0	14990
	Scenic	2000.0	2018.0	140	Neu	Benzin	5.5	24990
Seat	Ibiza	138000.0	2008.0	69	Gebraucht	Benzin	5.9	3200
Volkswagen	Caddy	156600.0	2007.0	105	Gebraucht	Diesel	6.9	2890
	Passat	100.0	2018.0	67	Gebraucht	Benzin	4.7	10240
	T6	26537.0	2017.0	204	Jahreswagen	Diesel	7.6	60990
	Tiguan	13189.0	2018.0	179	Gebraucht	Benzin	7.6	39930
	Touran	45000.0	2014.0	140	Gebraucht	Benzin	7.2	17000
Volvo	V50	260000.0	2008.0	109	Gebraucht	Diesel	5.0	3200
	XC60	42656.0	2017.0	190	Gebraucht	Diesel	4.7	32930
	XC90	111183.0	2015.0	224	Gebraucht	Diesel	5.8	40930

[35]:
```
df = pd.read_csv("daten/Auto.csv",sep=";",decimal=",")
df[["Hersteller","Verbrauch"]]
```

[35]:

	Hersteller	Verbrauch
0	BMW	4.5
1	Volvo	4.7
2	Volvo	5.8
3	Audi	5.2
4	Seat	5.9
5	Volkswagen	7.6
6	Volkswagen	7.2
7	Volkswagen	4.7
8	Audi	5.7
9	Volvo	5.0
10	Renault	5.5
11	Renault	5.0
12	Kia	5.5
13	Peugeot	NaN
14	BMW	4.5
15	Volkswagen	6.9
16	Fiat	5.1
17	Mazda	6.8
18	Mazda	6.3
19	Volkswagen	7.6

[36]:
```
df[["Hersteller","Verbrauch"]].plot(x="Hersteller")

plt.show ()
```


Verbrauch
7.5
7.0
6.5
6.0
5.5
5.0
4.5
BMW
Volkswagen
Renault
Volkswagen
Hersteller