

Generell

Die numerischen Datentypen sind gleichermaßen zu behandeln wie in den bekannten Programmiersprachen.

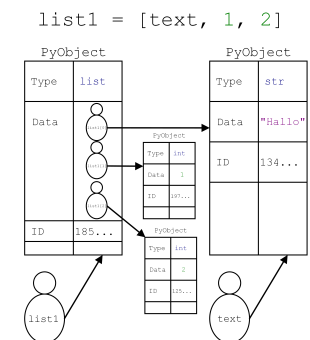
Datentypen

| Datentyp | Beschreibung | False-Wert |
|--------------------------------|--|-----------------|
| NoneType | Indikator für nichts | None |
| Numerische Datentypen | | |
| int | Ganze Zahlen | 0 |
| float | Gleitkommazahlen | 0.0 |
| bool | Boolesche Werte | False |
| complex | Komplexe Zahlen | 0 + 0j |
| Sequenzielle Datentypen | | |
| str | Zeichenketten oder Strings(unveränderlich) | ' ' |
| list | Listen(veränderlich) | [] |
| tuple | Tupel(unveränderlich) | () |
| bytes | Sequenz von Bytes(unveränderlich) | b' ' |
| bytearray | Sequenz von Bytes(veränderlich) | bytearray(b' ') |
| Mengen | | |
| set | Einmalig vorkommende Objekte | set() |
| frozenset | Wie set jedoch unveränderlich | frozenset() |
| Assoziative Datentypen | | |
| dict | Dictionary(veränderlich) | |

Operatoren

| Operator | Beschreibung |
|-----------|-----------------------|
| x // y | Ganzzahliger Quotient |
| x ** y | Potenzieren, x^y |
| +, -, ... | Übliche Operation |

Variablen



Casting

| Datentyp | Klasse | Direkt | Datentyp | Klasse | Direkt |
|-----------------|-------------|------------------|----------------|-----------|------------|
| Ganzzahl | int() | 3 | Gleitkommazahl | float() | 3.1415 |
| Boolescher Wert | bool() | True,False | Komplexe Zahl | complex() | 2 + 4j |
| String | str() | "HSR", 'OST' | Liste | list() | [1,2,3] |
| Menge | set() | 1,2,3 | Tupel | tuple() | (1,2,3) |
| Unver. Menge | frozenset() | frozenset(1,2,3) | Dictionary | dict() | , "Key": 1 |

Eingabe und Ausgabe

```
name = input("type your name:")
print("Hello", name)
strList = ["YES", "WE", "CAN"]
for w in strList:
    print(w, end="--")
```

In:

type your name:MFG GG OG

Out:

Hello MFG GG OG

YES-WE-CAN-

| Parameter | Beschreibung | Default |
|-----------------|--|------------|
| object(s) | Alle Objekte werden in String konvertiert | |
| sep='separator' | Separierung der Objekte | ' ' |
| end='end' | Letztes Zeichen des print-Befehl | '\n' |
| file | Objekt mit einer Write-Methode | sys.stdout |
| flush | Boolescher Wert für die Output-Überprüfung | False |

Listen

| | | |
|---------------------|-------------------------------------|------------------------------------|
| list.append(x) | list.extend(iterable) | list.remove(x) |
| list.clear() | list.index(x[, start[, end]]) → int | list.reverse() |
| list.copy() → list | list.insert(i, x) | list.sort(key=None, reverse=False) |
| list.count(x) → int | list.pop([i]) → object | |

Erzeugen von Listen

```
mylist = [1, 2, 3]
alist = list([4, 5, 6])
blist = alist.copy() # deep copy
print(f'mylist = {mylist}, alist = {alist}')
print(f'blist = {blist}')
```

Out:

mylist = [1, 2, 3], alist = [4, 5, 6]

Hinzufügen/Entfernen von Elementen

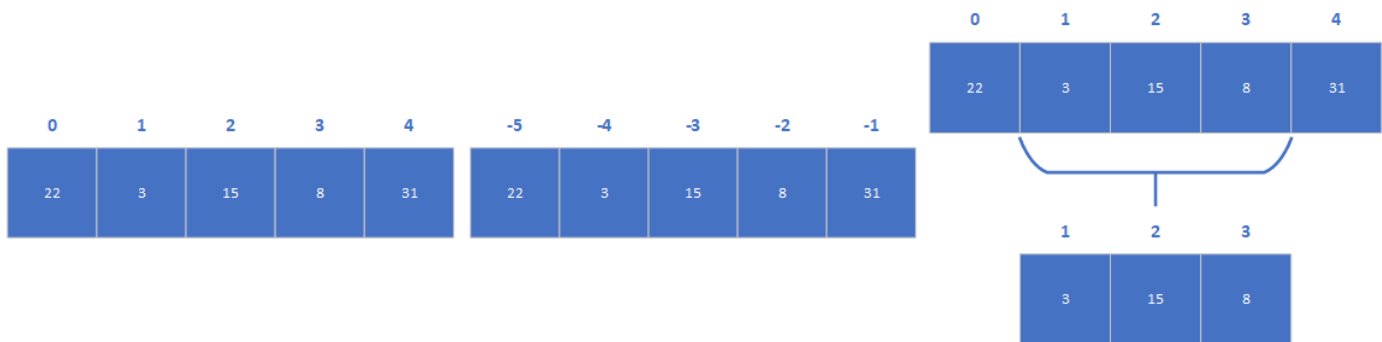
```
mylist = [1, 2]
mylist.append(33)
mylist.extend([11, 22])
print(f'#1 mylist = {mylist}')
mylist.pop()
mylist.pop(2)
mylist.remove(11)
print(f'#2 mylist = {mylist}')
mylist.clear()
print(f'#3 mylist = {mylist}')
```

Out:

```
#1 mylist = [1, 2, 33, 11, 22]
#2 mylist = [1, 2]
#3 mylist = []
```

Indexierung/Slicing

slice = lst[start:end:step] 'inklusive' start bis 'exklusiv' end



```
mylist = [22, 3, 15, 8, 31]
print(f'mylist[:] = {mylist[:]}')
print(f'mylist[1:4] = {mylist[1:4]}')
print(f'mylist[::2] = {mylist[::2]}')
print(f'mylist.index(3) = {mylist.index(3)}')
```

Out:

```
mylist[:] = [22, 3, 15, 8, 31]
mylist[1:4] = [3, 15, 8]
mylist[::2] = [22, 15, 31]
mylist.index(3) = 1
```

Sortieren

```
def myfunc(e):
    return len(e)

mylist = [4, 3, 5, 7, 3, 2]
strlist = ['BMW', 'Tesla', 'GM']
mylist.sort()
print('#1', mylist)
mylist.sort(reverse=True)
print('#2', mylist)
print('#3', mylist.count(2))
strlist.sort(key=myfunc)
print('#4', strlist)
```

Out:

```
#1 [2, 3, 3, 4, 5, 7]
#2 [7, 5, 4, 3, 3, 2]
#3 1
#4 ['GM', 'BMW', 'Tesla']
```

Tuple

Tuples sind ähnlich zu behandeln wie die Listen, nur dass sie nicht veränderbar sind.

`tuple.index(x)` `tuple.count(x)`

```
mytuplel = (1, "one", [0, 1])
print(mytuplel)
```

Out:

```
(1, 'one', [0, 1])
```

Set

| | | |
|--------------------------------|----------------------------------|--|
| set.add(elem) | set.intersection(*others) → set | set.remove(elem) |
| set.clear() | set.intersection_update(*others) | set.symmetric_difference(other) → set |
| set.copy() → set | set.isdisjoint(other) → bool | set.symmetric_difference_update(other) |
| set.difference(*others) → set | set.issubset(other) → bool | set.union(*others) → set |
| set.difference_update(*others) | set.issuperset(other) → bool | set.update(*others) |
| set.discard(elem) | set.pop() → object | |

Ein set kann ebenfalls wie eine Liste behandelt werden, wobei die Werte nur einmalig vorkommen können und sie nicht veränderbar sind. Weitere Werte können hinzugefügt oder entfernt werden. Sets sind ungeordnet, heisst, **die Reihenfolge der Elemente kann nicht vorhergesagt werden.**

Erzeugen

```
myset = {'I', 'love', 'swiss', 'cheese'}
aset = {'one', 'two', 'one', 'six'}
print(f'myset = {myset}')
print(f'aset = {aset}')
```

Out:

```
myset = 'I', 'cheese', 'love', 'swiss'
aset = 'six', 'two', 'one'
```

Basics

```
myset = {4, 12, 77, 120}
myset.add('elem')
myset.remove(120)
myset.pop()
myset.discard('nothing')
print(f'myset = {myset}')
aset = myset.copy()
myset.clear()
print(f'myset = {myset}')
print(f'aset = {aset}')
```

Out:

```
myset = 12, 77, 'elem'
myset = set()
aset = 'elem', 12, 77
```

Dictionary

| | |
|---|--|
| dict.clear() | dict.pop(key[, default]) → object |
| dict.copy() → dict | dict.popitem() → tuple |
| dict.fromkeys(iterable[, value]) → object | dict.setdefault(key[, default]) → object |
| dict.get(key[, default]) → object | dict.update([other]) |
| dict.items() → dict_items | dict.values() → dict_values |
| dict.keys() → dict_keys | |

Ein Dictionary ist ein geordneter Datentyp um Wertepaare zu speichern. Dieser Datentyp erlaubt keine Duplikate!

Basics

```
cb300 = {
    'brand': 'honda',
    'color': 'red',
    'power': (31, 'hp')
}
update = {'category': 'naked'}
cb300['mileage[km]'] = 3148
cb300['mileage[km]'] = 4399
cb300.update(update)
cb500 = cb300.copy()
cb500['power'] = (48, 'hp')
cb500['color'] = 'grey'
print(cb300)
print(cb500)
```

Out:

```
{'brand': 'honda', 'color': 'red', 'power': (31, 'hp'),
'mileage[km]': 4399, 'category': 'naked'}
{'brand': 'honda', 'color': 'red', 'power': (48, 'hp'),
'mileage[km]': 4399, 'category': 'naked'}
```

Entfernen/Löschen

```
cb300 = {
    'brand':    'honda',
    'color':    'red',
    'power':    (31, 'hp')
}
print(cb300.pop('color', 'raw'))
print(cb300.pop('color', 'raw'))
print(cb300.popitem()) #removes last added
                        item
cb300.clear()
print(cb300)
```

Out:

```
red
raw
('power', (31, 'hp'))
{}
```

Abfragen/Hinzufügen

```
cb300 = {
    'brand':    'honda',
    'color':    'red',
    'power':    (31, 'hp')
}
keys = ('year', 'weight')
values = 0
print(cb300.items())
print(cb300.keys())
print(cb300.values())
print(cb300.get('color', 'raw'))
print(cb300.get('mileage', 0))
print(cb300.setdefault('color', 'black'))
cb300.update(dict.fromkeys(keys, values))
print(cb300)
```

Out:

```
dict_items([('brand', 'honda'), ('color', 'red'),
('power', (31, 'hp'))])
dict_keys(['brand', 'color', 'power'])
dict_values(['honda', 'red', (31, 'hp')])
red
0
red
{'brand': 'honda', 'color': 'red', 'power': (31, 'hp'),
'year': 0, 'weight': 0}
```

Strings

```
string1 = "python"
string2 = """Mehrzeiliger
String"""
t = string1[2]
t = string1[-4]
```

```
# F-String example
fruit = 'apple'
color = 'red'
print(f'The fruit {fruit} has
      the color {color}')
```

Matplotlib

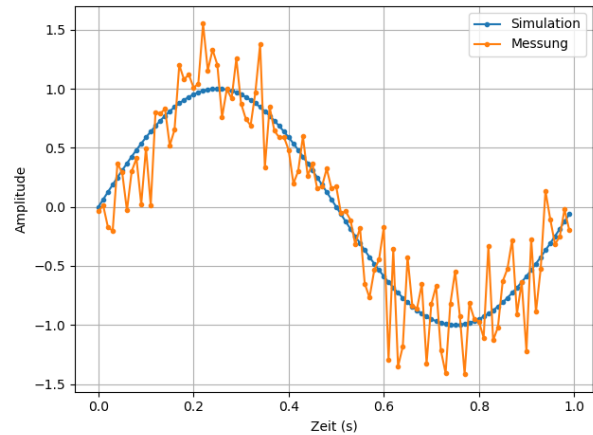
| | |
|---|---------------------------------------|
| Creation | |
| <code>plt.figure(num=None, dpi=None, facecolor=None, edgecolor=None, frameon=True, FigureClass=plt.figure.Figure, clear=False, **kwargs)</code> | <code>→ Figure</code> |
| <code>plt.subplot(*args, **kwargs)</code> | <code>→ AxesSubplot</code> |
| <code>plt.subplots(nrows=1, ncols=1, *, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw)</code> | <code>Figure, → axes.Axes</code> |
| <code>plt.twinx(ax=None)</code> | <code>→ AxesSubplot</code> |
| <code>plt.twiny(ax=None)</code> | <code>→ AxesSubplot</code> |
| <code>plt.tight_layout(*, pad=1.08, h_pad=None, w_pad=None, rect=None)</code> | |
| <code>plt.savefig(*args, **kwargs)</code> | |
| <code>plt.show(*, block=None)</code> | |
| Drawing | |
| <code>plt.annotate(text, xy, *args, **kwargs)</code> | <code>→ Annotation</code> |
| <code>plt.arrow(x, y, dx, dy, **kwargs)</code> | <code>→ FancyArrow</code> |
| <code>plt.contour(*args, data=None, **kwargs)</code> | <code>→ QuadContourSet</code> |
| <code>plt.contourf(*args, data=None, **kwargs)</code> | <code>→ QuadContourSet</code> |
| <code>plt.loglog(*args, **kwargs)</code> | <code>→ list of Line2D objects</code> |
| <code>plt.plot(*args, scalex=True, scaley=True, data=None, **kwargs)</code> | <code>→ list of Line2D objects</code> |
| <code>plt.scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=deprecated parameter, edgecolors=None, *, plotnonfinite=False, data=None, **kwargs)</code> | <code>→ PathCollection</code> |
| <code>plt.semilogx(*args, **kwargs)</code> | <code>→ list of Line2D objects</code> |
| <code>plt.semilogy(*args, **kwargs)</code> | <code>→ list of Line2D objects</code> |
| <code>plt.stem(*args, linefmt=None, markerfmt=None, basefmt=None, bottom=0, label=None, use_line_collection=True, data=None)</code> | <code>→ StemContainer</code> |
| <code>plt.step(x, y, *args, where="pre", data=None, **kwargs)</code> | <code>→ list of Line2D objects</code> |
| <code>plt.streamplot(x, y, u, v, density=1, linewidth=None, color=None, cmap=None, norm=None, arrowsize=1, arrowstyle="→", minlength=0.1, transform=None, zorder=None, start_points=None, maxlength=4.0, integration_direction="both", *, data=None)</code> | <code>→ StreamplotSet</code> |
| <code>plt.text(x, y, s, fontdict=None, **kwargs)</code> | <code>→ Text</code> |
| Decoration | |
| <code>plt.axis(*args, emit=True, **kwargs)</code> | <code>→ Annotation</code> |
| <code>plt.axhline(y=0, xmin=0, xmax=1, **kwargs)</code> | <code>→ Line2D</code> |
| <code>plt.axvline(x=0, ymin=0, ymax=1, **kwargs)</code> | <code>→ Line2D</code> |
| <code>plt.colorbar(mappable=None, cax=None, ax=None, **kw)</code> | <code>→ Colorbar</code> |
| <code>plt.grid(b=None, which="major", axis="both", **kwargs)</code> | |
| <code>plt.legend(*args, **kwargs)</code> | |
| <code>plt.margins(*margins, x=None, y=None, tight=True)</code> | <code>→ tuple</code> |
| <code>plt.suptitle(t, **kwargs)</code> | <code>Text</code> |
| <code>plt.title(label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs)</code> | <code>→ Text</code> |
| <code>plt.xlabel(xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)</code> | |
| <code>plt.xlim(*args, **kwargs)</code> | <code>→ tuple</code> |
| <code>plt.xticks(ticks=None, labels=None, **kwargs)</code> | <code>→ tuple</code> |
| <code>plt.ylabel(ylabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)</code> | |
| <code>plt.ylim(*args, **kwargs)</code> | <code>→ tuple</code> |
| <code>plt.yticks(ticks=None, labels=None, **kwargs)</code> | <code>→ tuple</code> |
| Decoration OOP | |
| <code>Axes.set_title(self, label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs)</code> | <code>→ Text</code> |
| <code>Axes.set_xlabel(self, xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)</code> | |
| <code>Axes.set_xlim(self, left=None, right=None, emit=True, auto=False, *, xmin=None, xmax=None)</code> | <code>→ tuple</code> |
| <code>Axes.set_xticks(self, ticks, *, minor=False)</code> | <code>→ tuple</code> |
| <code>Axes.set_ylabel(self, ylabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)</code> | |
| <code>Axes.set_ylim(self, bottom=None, top=None, emit=True, auto=False, *, ymin=None, ymax=None)</code> | <code>→ tuple</code> |
| <code>Axes.set_yticks(self, ticks, *, minor=False)</code> | <code>→ tuple</code> |

| Linestyle | Marker | Color |
|-----------------|---------------------------|---|
| '-', 'solid' | 'o'=point 'o'=circle | 'v'=triangle_down 'b','blue' 'g','green' 'r','red' |
| ':', 'dotted' | 's'=square 'p'=pentagon | '^'=triangle_right 'c','cyan' 'm','magenta' 'k','black' |
| '--', 'dashed' | '*' =star 'd','D'=diamond | '^'=triangle_up 'w','white' 'y','yellow' '#0F0F0F' |
| '-.', 'dashdot' | 'x','X'=x '+' , 'P'=plus | 'i'=triangle_left (0.0, 0.5, 1.0) |

Einfaches Beispiel

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

# signale definieren
t = np.arange(100)/100
s1 = np.sin(2*np.pi*t)
s2 = s1 + np.random.randn(*s1.shape)/4
# Figure und Axis erstellen
plt.figure()
# Signale plotten
plt.plot(t, s1, '-.', label='Simulation')
plt.plot(t, s2, '-.', label='Messung')
# Achsen beschriften
plt.xlabel('Zeit (s)')
plt.ylabel('Amplitude')
# Hilfslinien aktivieren
plt.grid(True)
# Legende hinzufuegen
plt.legend()
# Weisser Rand minimieren
plt.tight_layout()
# Diagramm abspeichern
plt.savefig('diagramm.svg')
```

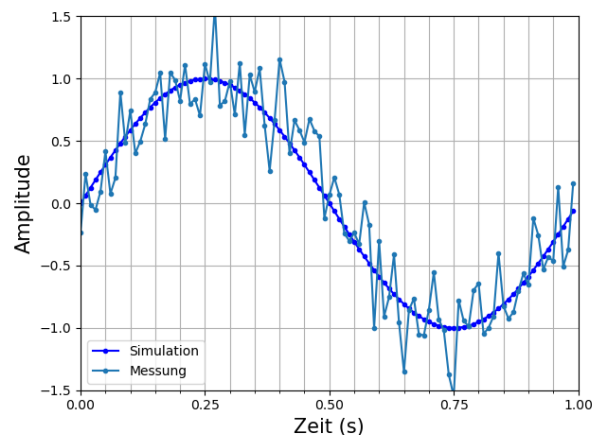


OOP Beispiel

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import (
    MultipleLocator, AutoMinorLocator)

t = np.arange(100)/100
s1 = np.sin(2*np.pi*t)
s2 = s1 + np.random.randn(*s1.shape)/4
# Figure und Axis erstellen
fig, ax = plt.subplots()
# Signale plotten
ax.plot(t, s1, linestyle='-', marker='.',
        color="blue", label='Simulation')
# ax.plot(t, s1, '-.', label='Simulation')
ax.plot(t, s2, '-.', label='Messung')
# Achsen beschriften
ax.set_xlabel('Zeit (s)', fontsize=15)
ax.set_ylabel('Amplitude', fontsize=15)
# Achsen limitieren
ax.set_ylim(-1.5, 1.5)
# ax.set_xticks(np.arange(0, 1.1, step=0.1))
ax.set_xlim(0, 1)

ax.xaxis.set_major_locator(MultipleLocator(
    0.25))
ax.xaxis.set_minor_locator(MultipleLocator(
    0.05))
# Hilfslinien aktivieren
# ax.grid(True)
ax.grid(which="both")
# Legende hinzufuegen
ax.legend(loc="lower left")
# Weisser Rand minimieren
fig.tight_layout()
```



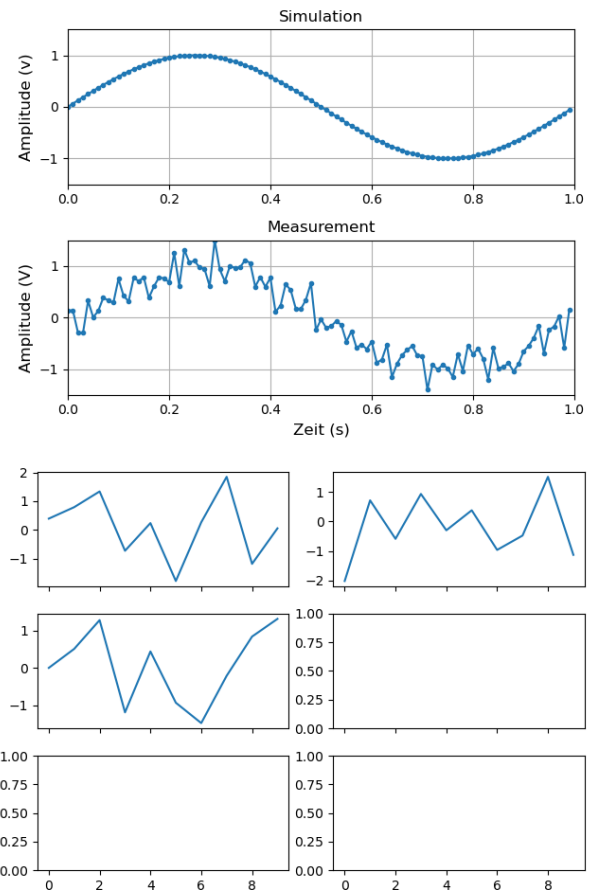
Subplots

```

import numpy as np
import matplotlib.pyplot as plt

t = np.arange(100)/100
s1 = np.sin(2*np.pi*t)
s2 = s1 + np.random.randn(*s1.shape)/4
# Figure und Axis erstellen
fig = plt.figure()
ax1 = plt.subplot(2, 1, 1)
ax2 = plt.subplot(2, 1, 2, sharex=ax1)
# Signale plotten
ax1.plot(t, s1, '.-', label='Simulation')
ax2.plot(t, s2, '.-', label='Simulation')
# Achsen beschriften
# ax1.set_xlabel('Zeit (s)', fontsize=12)
ax1.set_ylabel('Amplitude (v)', fontsize=12)
ax2.set_xlabel('Zeit (s)', fontsize=12)
ax2.set_ylabel('Amplitude (V)', fontsize=12)
# Achsen limitieren
ax1.set_ylim(-1.5, 1.5)
ax2.set_ylim(-1.5, 1.5)
ax1.set_xlim(0, 1)
# Subplots beschriften
ax1.set_title("Simulation")
ax2.set_title("Measurement")
# Hilfslinien aktivieren
ax1.grid(True)
ax2.grid(True)
# Weisser Rand minimieren
fig.tight_layout()
#####
# Multiple subplots
fig, axes = plt.subplots(3, 2, sharex=True)
axes[0][0].plot(np.random.randn(10))
axes[0][1].plot(np.random.randn(10))
axes[1][0].plot(np.random.randn(10))
fig.tight_layout()

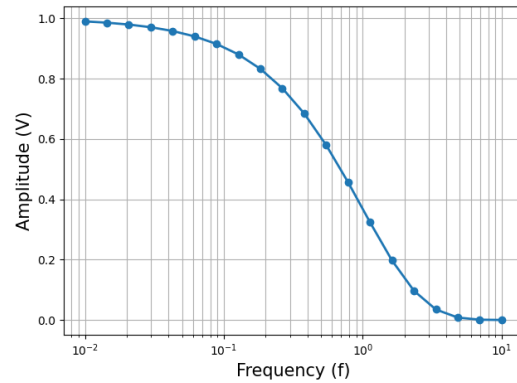
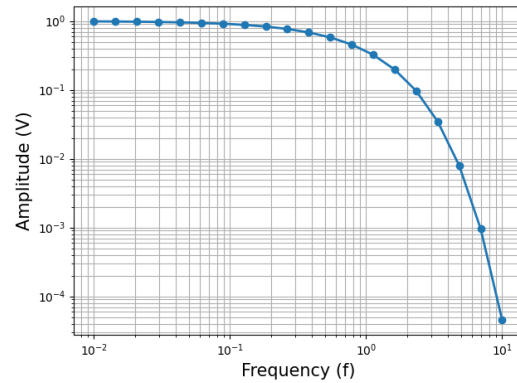
```



logarithmische Darstellung

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

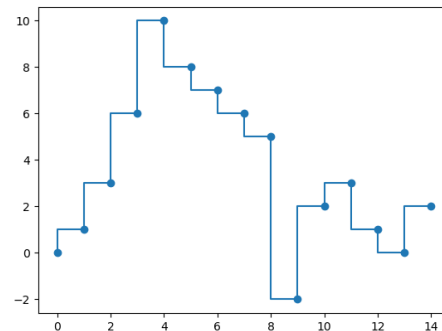
x = np.logspace(-2, 1, 20)
s = np.exp(-x)
# loglog-----
fig, ax = plt.subplots()
ax.loglog(x, s, '-o', linewidth=2)
ax.grid(which='major')
ax.grid(which='minor')
ax.set_xlabel('Frequency (f)', fontsize=15)
ax.set_ylabel('Amplitude (V)', fontsize=15)
fig.tight_layout()
# semilogx-----
fig, ax = plt.subplots()
ax.semilogx(x, s, '-o', linewidth=2)
ax.grid(which='major')
ax.grid(which='minor')
ax.set_xlabel('Frequency (f)', fontsize=15)
ax.set_ylabel('Amplitude (V)', fontsize=15)
fig.tight_layout()
# semilogy-----
```



step

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

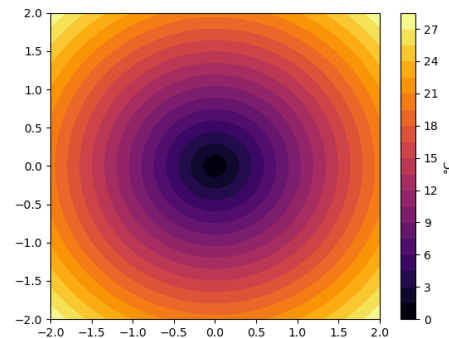
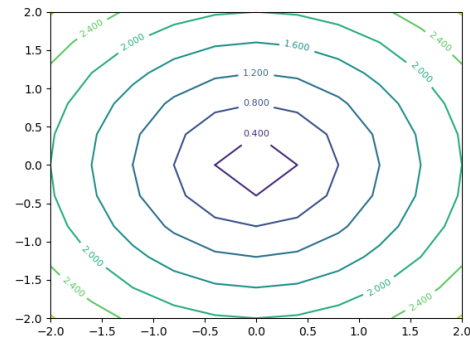
x = np.logspace(-2, 1, 20)
s = np.exp(-x)
fig, ax = plt.subplots()
x = np.arange(15)
y = np.array([0, 1, 3, 6, 10, 8, 7, 6, 5, -2,
              2, 3, 1, 0, 2])
ax.step(x, y, '-o', where='pre');
```



contour/contourf

```
import numpy as np
import matplotlib.pyplot as plt
# contour-----
x = y = np.linspace(-2, 2, 11)
X, Y = np.meshgrid(x, y)
Z = np.sqrt(X**2 + Y**2)
fig, ax = plt.subplots()
cont = ax.contour(X, Y, Z);
ax.clabel(cont, inline=True, fontsize=8);
# contourf-----
x = y = np.linspace(-2, 2, 100)
X, Y = np.meshgrid(x, y)
Z = np.sqrt(X**2 + Y**2)*10

fig, ax = plt.subplots()
cont = ax.contourf(X, Y, Z, 20, cmap='inferno');
cbar = fig.colorbar(cont);
cbar.ax.set_ylabel('C');
```

**hist**

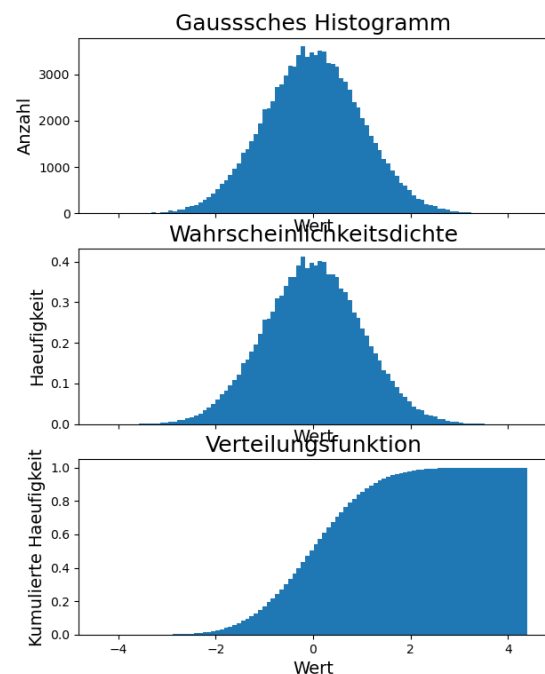
```
import numpy as np
import matplotlib.pyplot as plt

N = 100000
data = np.random.randn(N)

fig, axes = plt.subplots(3, sharex=True)
axes[0].hist(data, bins=100)
axes[0].set_title('Gausssches Histogramm',
    fontsize=18)
axes[0].set_xlabel('Wert', fontsize=14)
axes[0].set_ylabel('Anzahl', fontsize=14);

axes[1].hist(data, bins=100, density = True)
axes[1].set_title('Wahrscheinlichkeitsdichte',
    fontsize=18)
axes[1].set_xlabel('Wert', fontsize=14)
axes[1].set_ylabel('Haeufigkeit', fontsize=14);

axes[2].hist(data, bins=100, density = True,
    cumulative=True)
axes[2].set_title('Verteilungsfunktion',
    fontsize=18)
axes[2].set_xlabel('Wert', fontsize=14)
axes[2].set_ylabel('Kumulierte Haeufigkeit',
    fontsize=14);
```



scatter

```

import numpy as np
import matplotlib.pyplot as plt

x = [160, 162, 180, 182, 172, 172, 160, 178,
     175, 190, 175, 168, 165]
y = [50, 58, 80, 85, 57, 65, 45, 80, 78, 100,
     60, 55, 55]
# test = np.loadtxt("body_dimensions_data.txt",
#                   usecols=(0))
data = np.loadtxt("body_dimensions_data.txt")
height = data[:, -2]
weight = data[:, -3]
s = data[:, -1]
male = s == 1
female = s == 0

fig, axes = plt.subplots(nrows=2)
scat_male = axes[0].scatter(height[male],
                             weight[male], color='b')
scat_female = axes[0].scatter(height[female],
                               weight[female], color='r')
axes[0].set_xlabel('Groesse (cm)', fontsize=14)
axes[0].set_ylabel('Gewicht (kg)', fontsize=14)
axes[0].legend((scat_male, scat_female), ('Male', 'Female'),
               fontsize=8)
# other example of scatter-----
from sklearn.datasets import load_iris
iris = load_iris()
features = iris.data.T
scat = axes[1].scatter(features[0], features[1],
                       alpha=0.2, s=100*features[3],
                       c=iris.target, cmap='viridis')
axes[1].set_xlabel(iris.feature_names[0],
                   fontsize='16')
axes[1].set_ylabel(iris.feature_names[1],
                   fontsize='16')
axes[1].legend(handles=scat.legend_elements()[0],
               labels=list(iris.target_names),
               loc='upper left')

```

