```
# coding: utf-8

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

1 Lektion 8

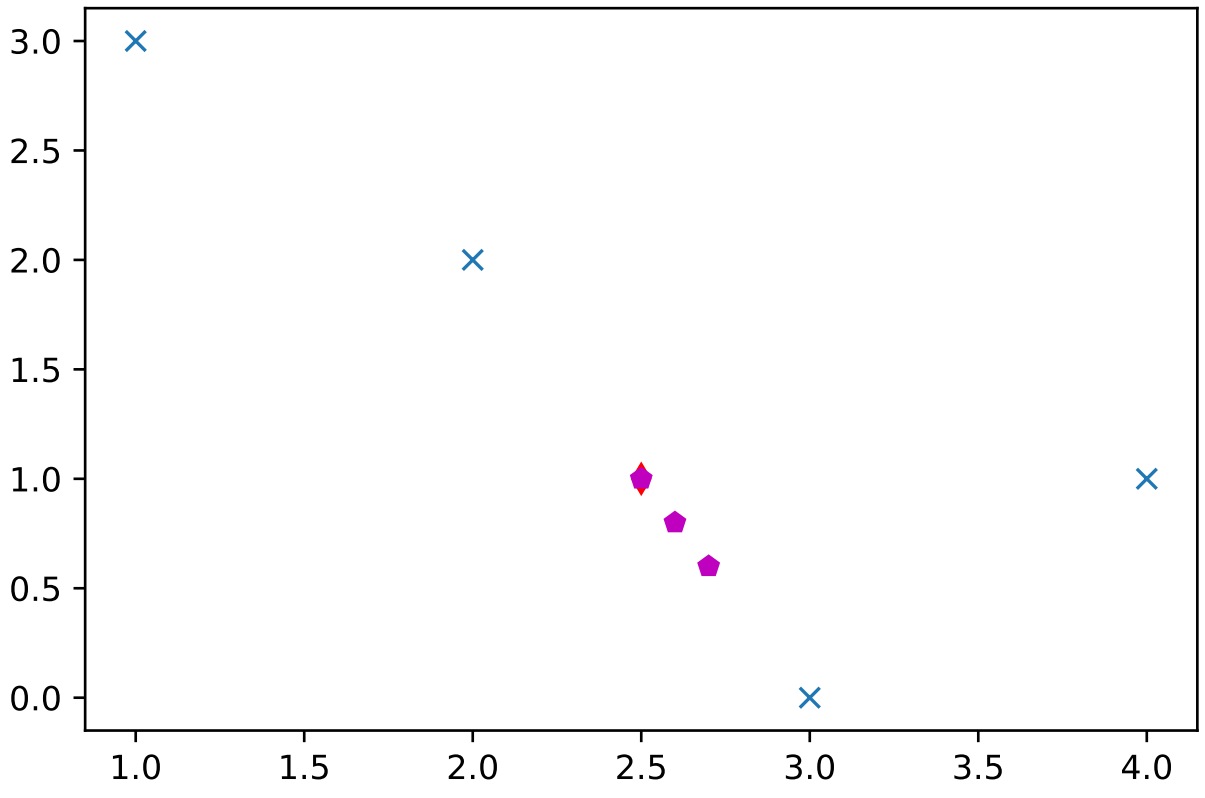
1.1 Zeichnen und Interpolation

```
a = [1, 2, 3, 4]
b = [3, 2, 0, 1]
```

Polygonzug zeichnen

```
fig = plt.figure(1)          # erzeugt ein Fenster fuer Graphik
ax = fig.add_subplot(111)    # 'ax' ist das Koordinatensystem
ax.plot(a, b,'x');

bb = np.interp(2.5, a, b)
ax.plot(2.5, bb, 'rd');

aas = np.array([2.5, 2.6, 2.7])
bbs = np.interp(aas, a, b)
ax.plot(aas, bbs, 'mp');
```

```
[<matplotlib.lines.Line2D at 0x7faf18459a20>]
```



Verschönerung

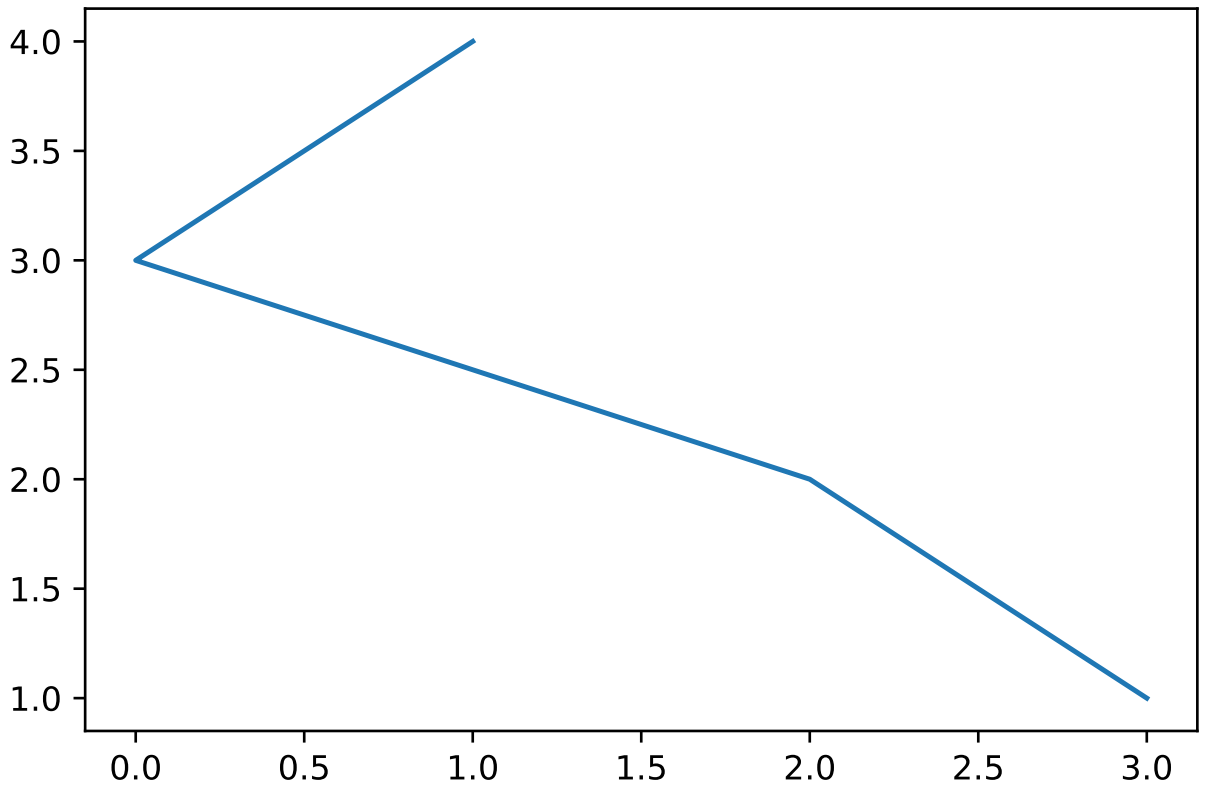
```
ax.set_xlim([1, 3])
ax.set_xlabel('x Achse')
ax.set_ylabel('meine Funktion')
ax.axis([-1, 4, 0, 5]);
```

```
[-1, 4, 0, 5]
```

x und y Achse vertauscht

```
fig = plt.figure(2)
plt.plot(b, a)
```

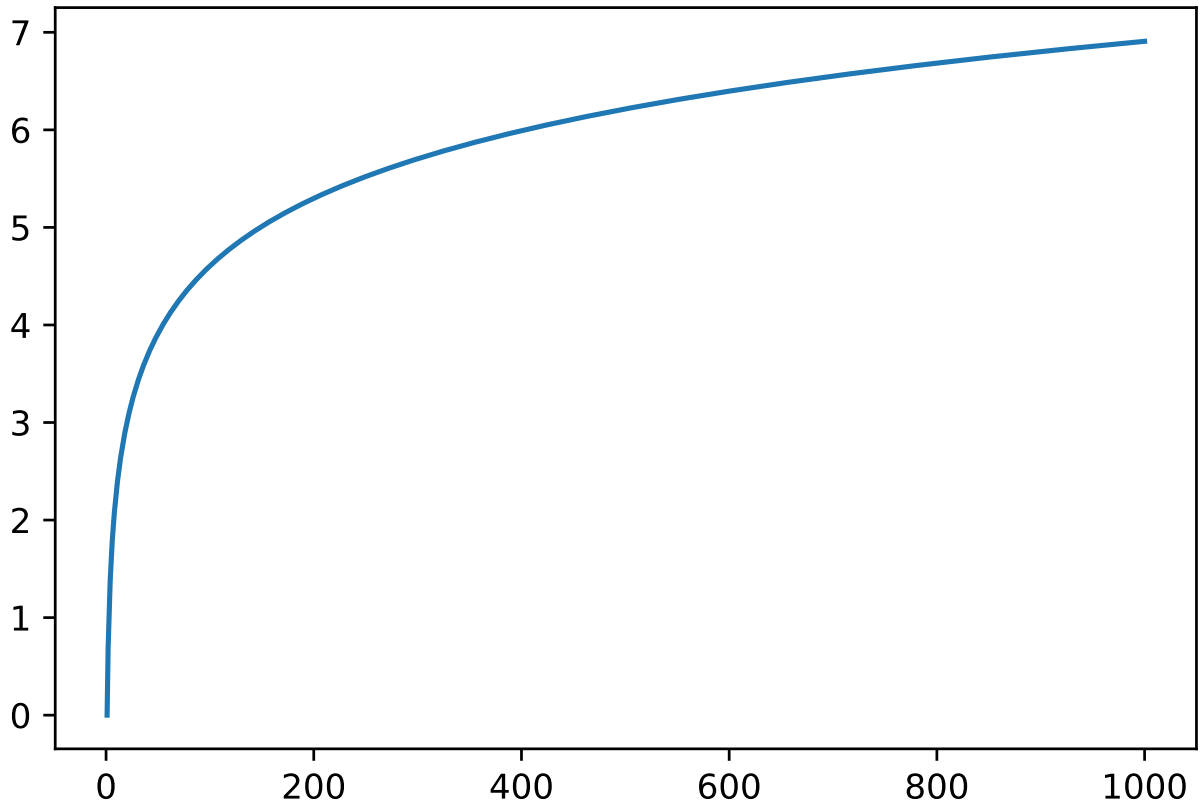
```
[<matplotlib.lines.Line2D at 0x7faf181de358>]
```



1.2 Logarithmisch skalierte Achsen

```
x = np.arange(1000) + 1;
y = np.log(x);
fig = plt.figure()
plt.plot(x, y);
```

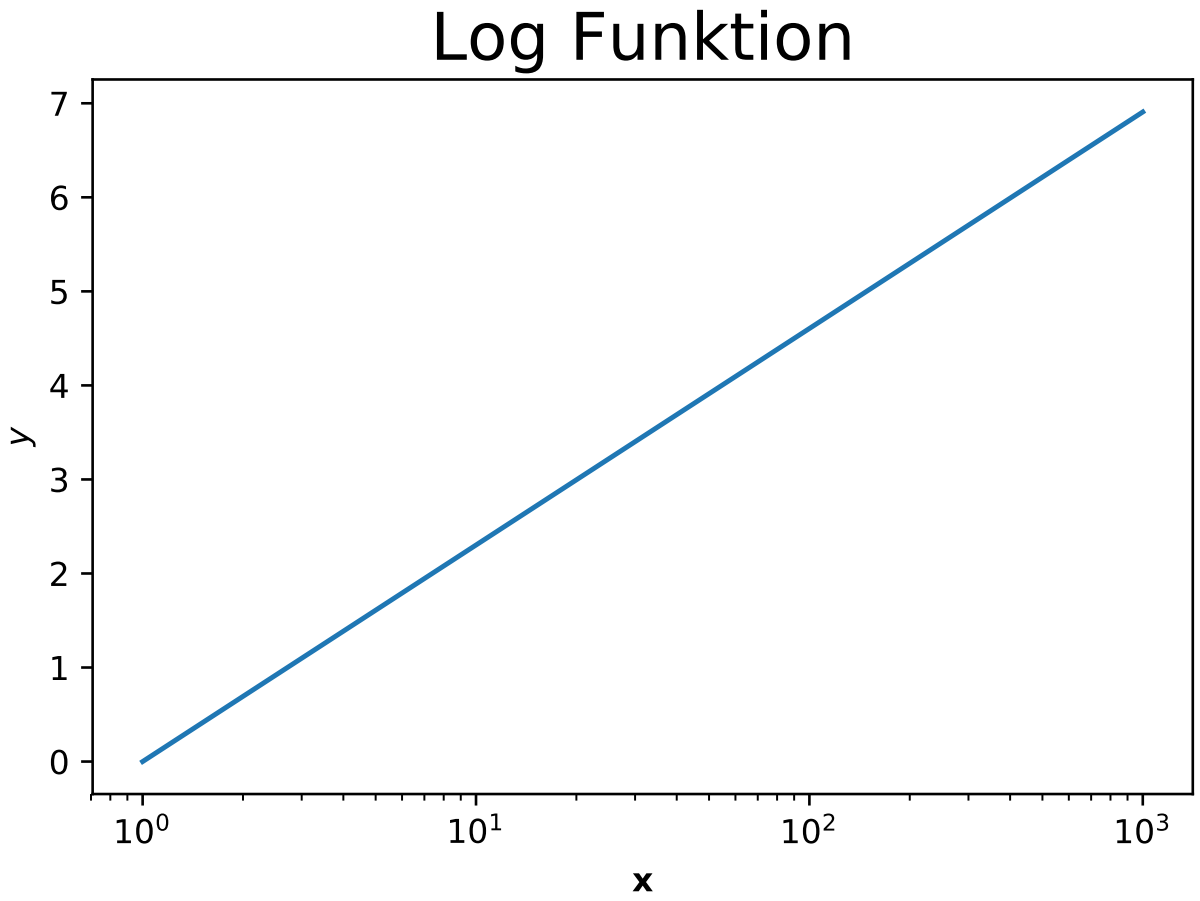
```
[<matplotlib.lines.Line2D at 0x7faf18145ef0>]
```



Semilogx

```
fig = plt.figure(3)
ax = plt.axes(xlabel="x", ylabel="y", title="Log Funktion")
ax.title.set_fontsize(20);
ax.xaxis.label.set_fontweight("bold");
ax.yaxis.label.set_fontstyle("italic");
ax.semilogx(x, y);
```

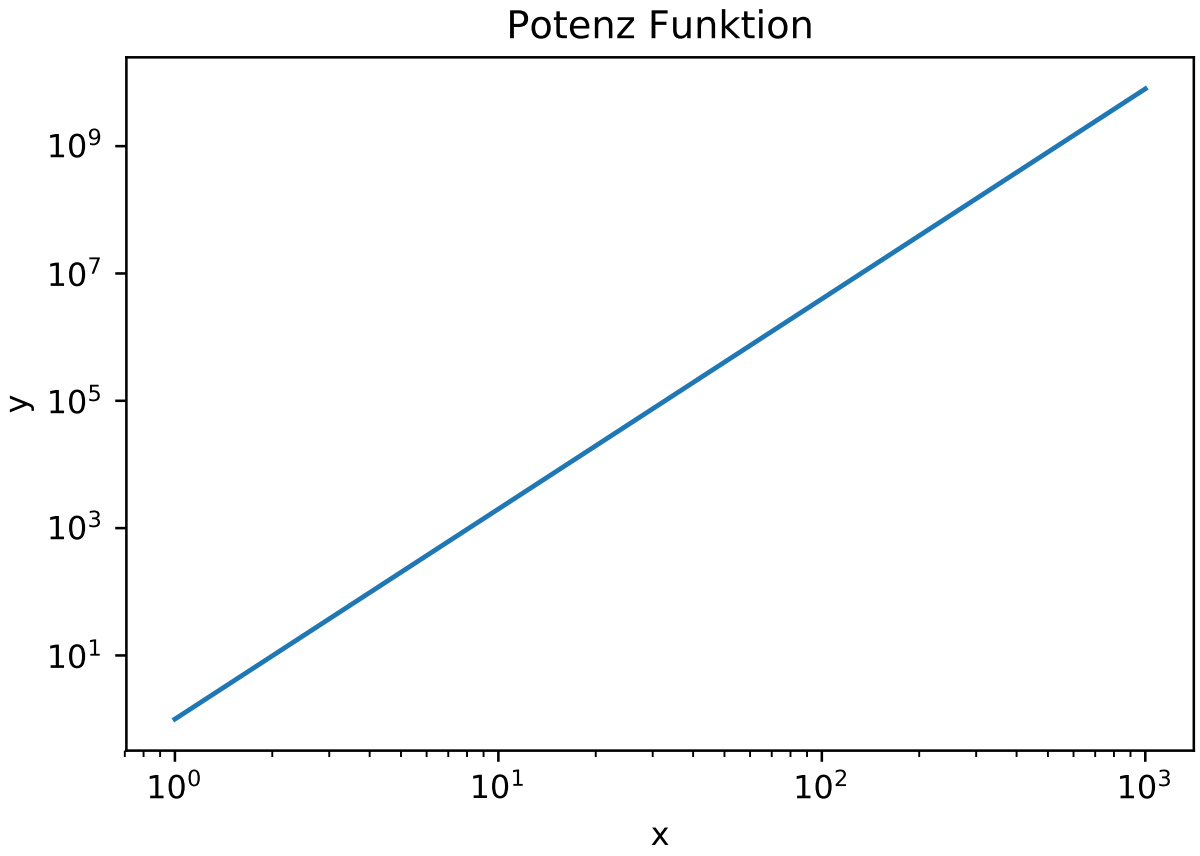
```
[<matplotlib.lines.Line2D at 0x7faf18122710>]
```



Loglog

```
y = x**3.3
plt.figure(13)
plt.axes(xlabel="x", ylabel="y", title="Potenz Funktion")
plt.loglog(x,y)
```

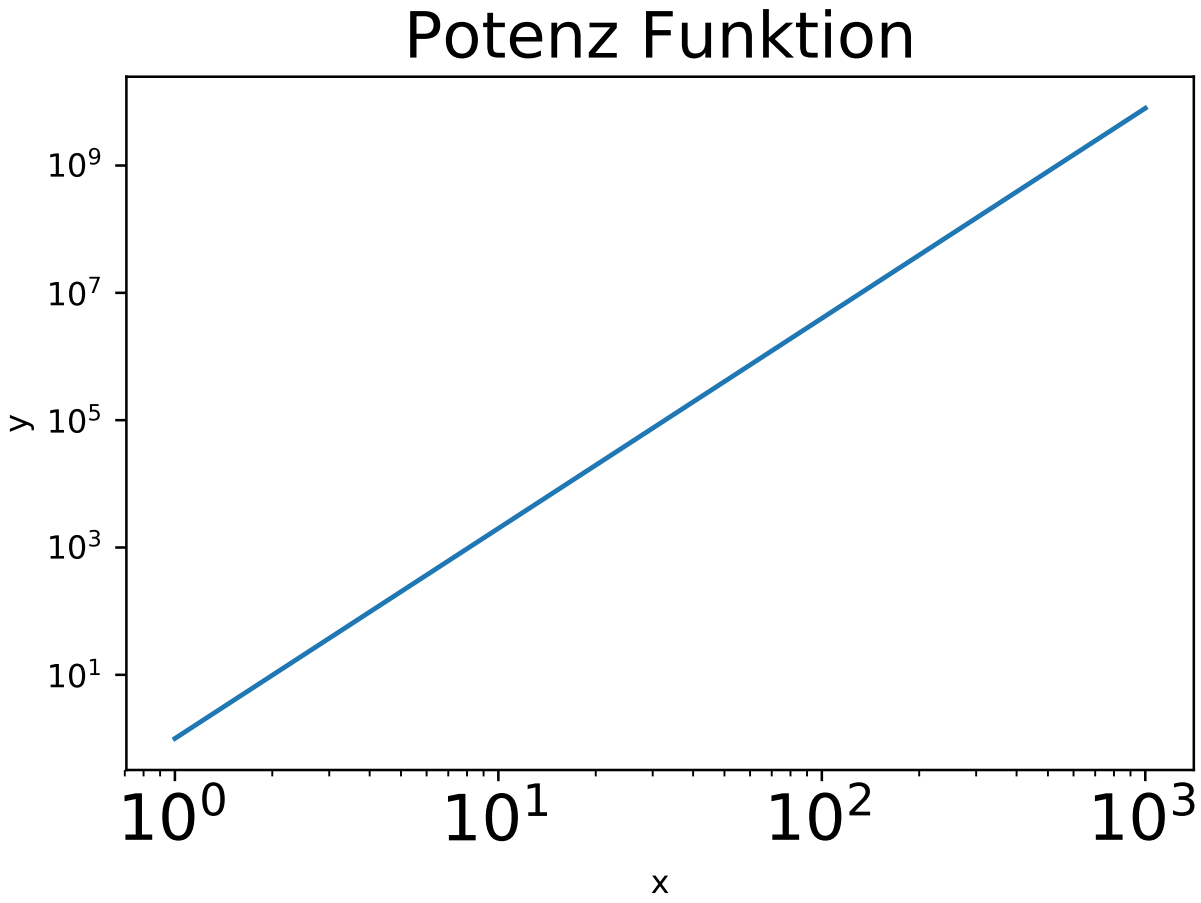
```
[<matplotlib.lines.Line2D at 0x7faf0ba334e0>]
```



Alternativ

```
plt.figure(111)
plt.xlabel("x")
plt.ylabel("y")
plt.title("Potenz Funktion",fontsize=20)
plt.xticks(fontsize=20)
plt.loglog(x,y);
```

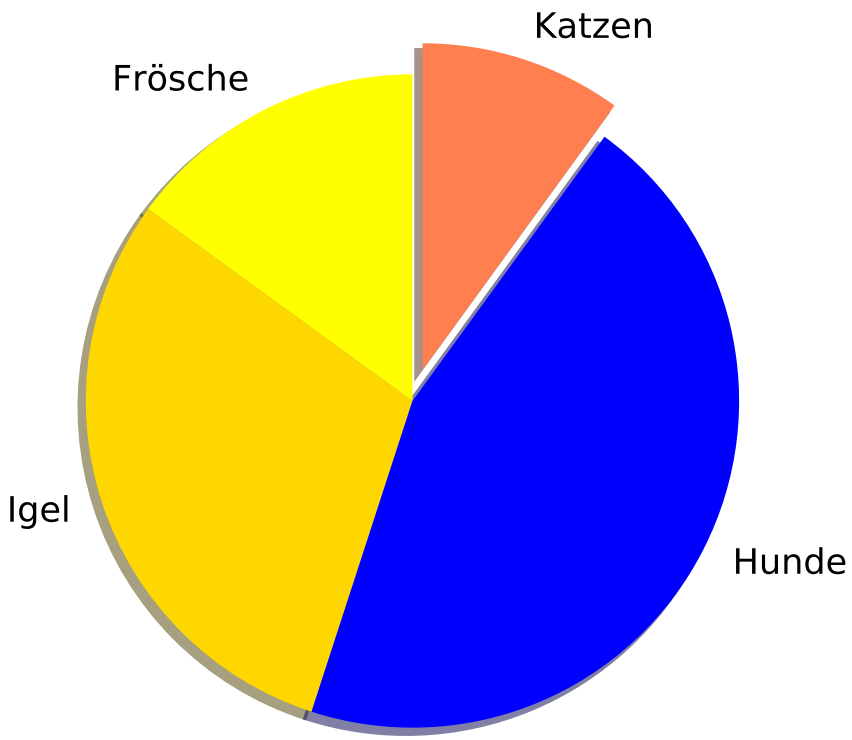
```
[<matplotlib.lines.Line2D at 0x7faf18107c50>]
```



1.3 Tortengraphik

```
tiere    = ["Frösche", "Igel", "Hunde", "Katzen" ];

anzahlen = [15,         30,     45,      10      ];

farben   = ["yellow",  "gold", "blue", "coral"  ];

fig = plt.figure(20)
plt.pie(anzahlen, labels=tiere, colors=farben, shadow=True,
        startangle=90, explode=[ 0, 0, 0, 0.1 ]);
plt.axis("equal");
```

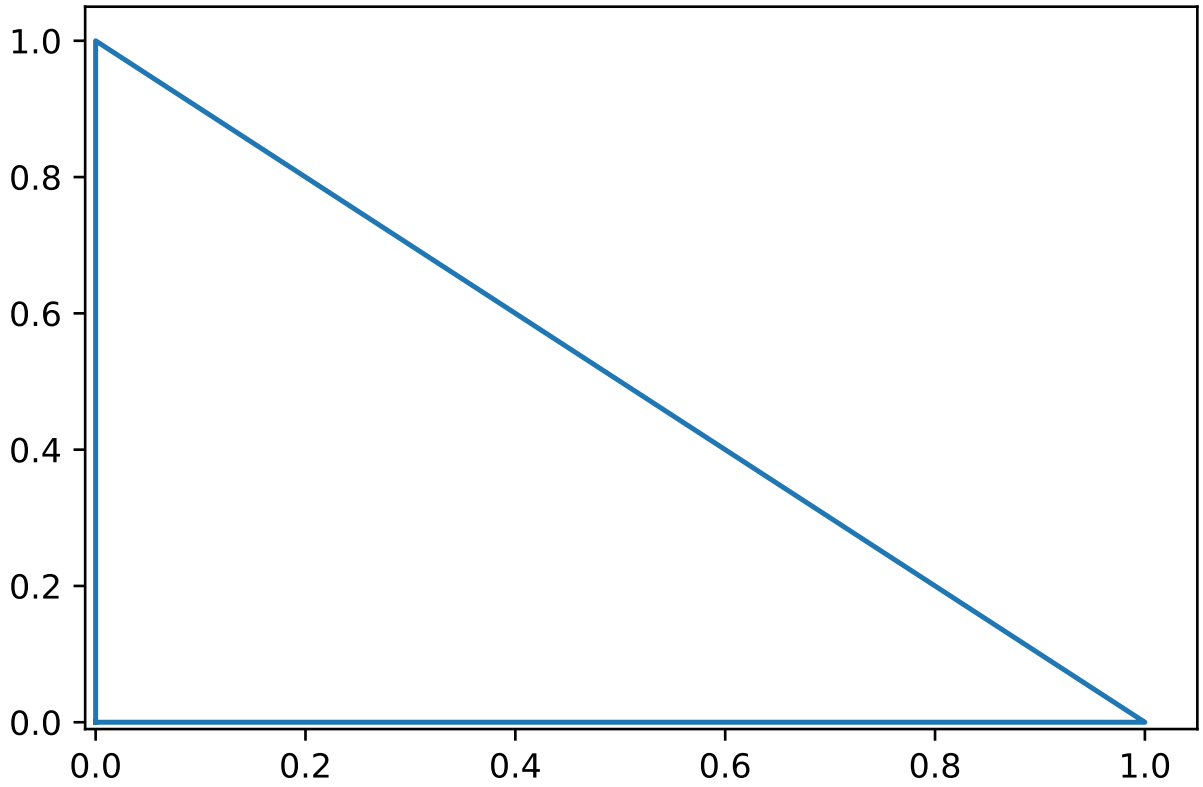
```
(-1.1143898348979537,
 1.1299557954617603,
 -1.1306021143878777,
 1.2010917422357943)
```



1.4 Wiederholung aus Lektion 5

```
plt.figure(30)
dreieck = [[0,0], [1,0], [0,1], [0,0]]
plt.plot([0,1,0,0], [0,0,1,0])
plt.axis(xmin=-.01,ymin=-.01);
```

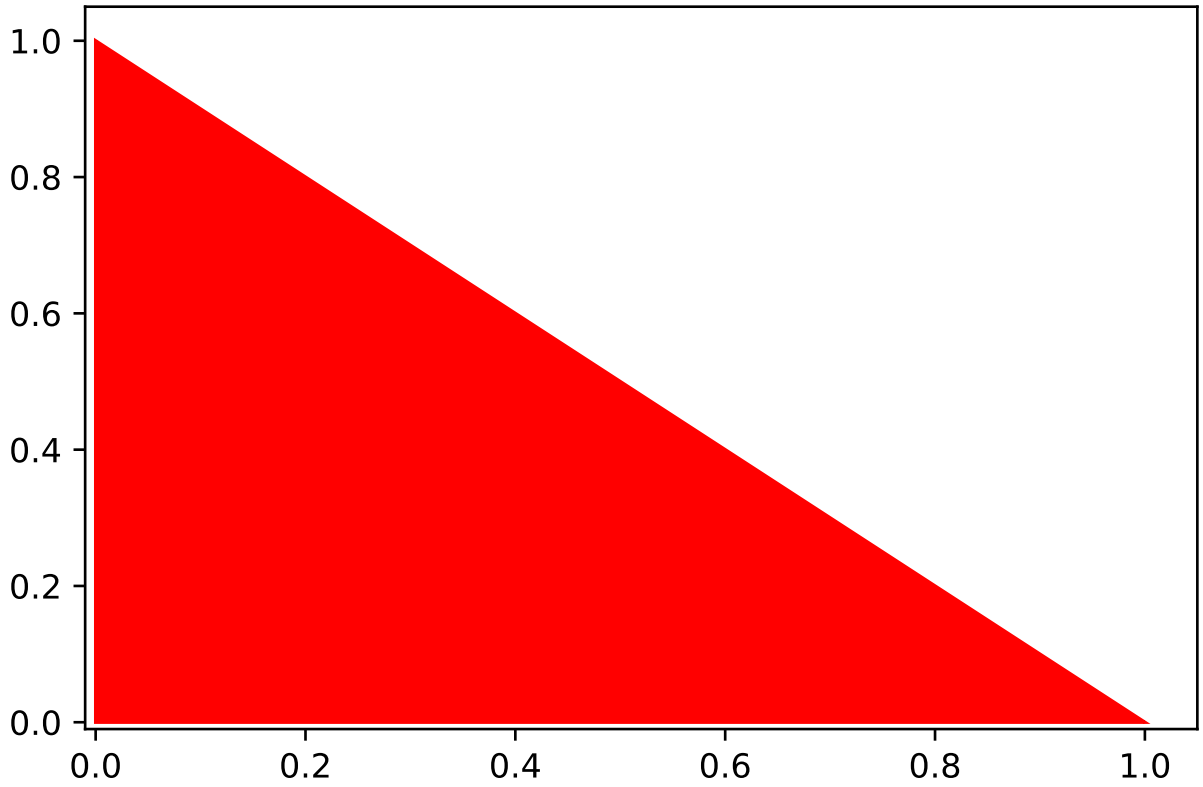
```
(-0.01, 1.05, -0.01, 1.05)
```



Ausgefüllte Kurve

```
plt.figure(31)
dreieck = [[0,0], [1,0], [0,1], [0,0]]
plt.fill([0,1,0,0], [0,0,1,0],color='red')
plt.axis(xmin=-.01,ymin=-.01);
```

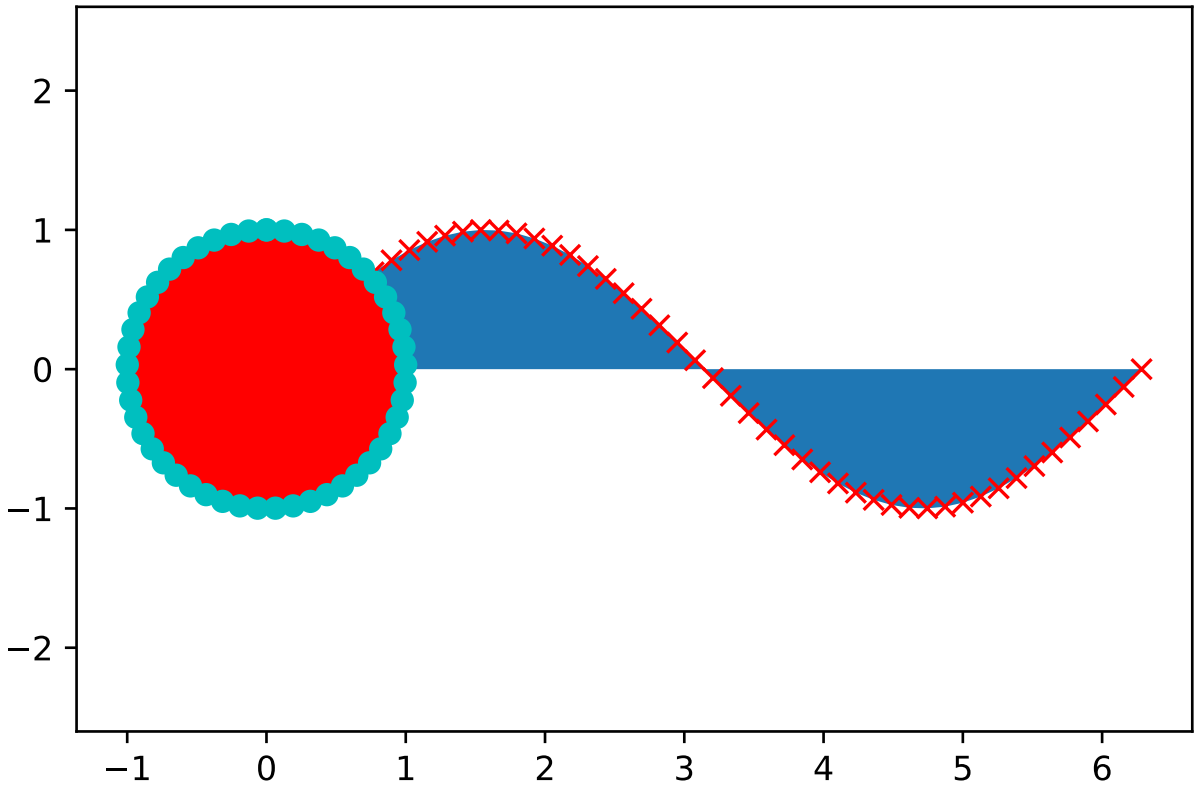
```
(-0.01, 1.05, -0.01, 1.05)
```



Ausgefüllte Kurve

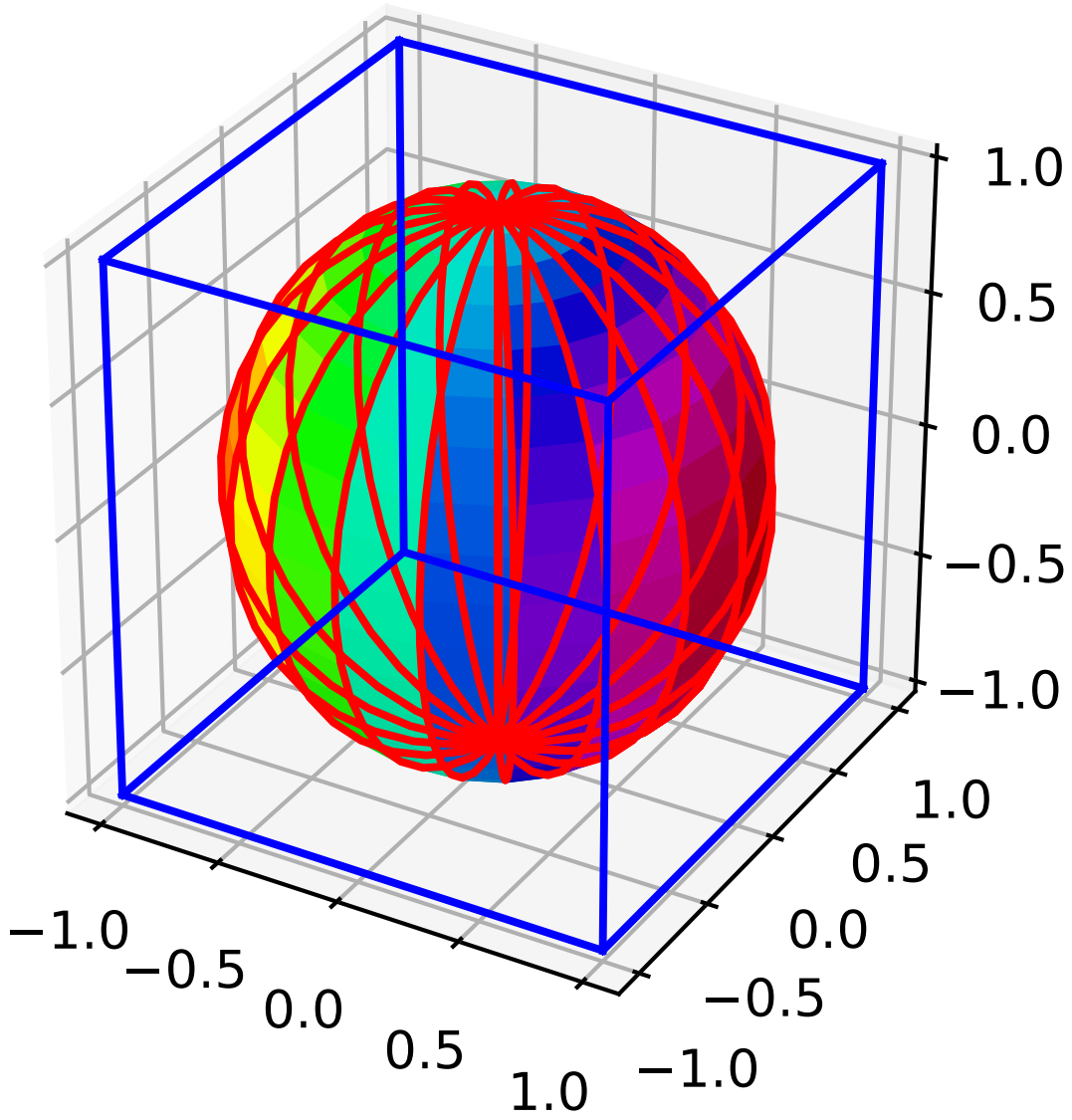
```
x = np.linspace(0,2*np.pi)
y1 = np.sin(x)
y2 = np.cos(x)

plt.figure(32)
plt.fill(x,y1)
plt.plot(x,y1,'xr')
plt.fill(y1,y2,'r')
plt.plot(y1,y2,'co')
plt.axis('equal')
```

```
(-1.3636197923697015, 6.6473188833486, -1.0994605270107223,
1.0999743108100344)
```



1.5 3D Graphik

```
from mpl_toolkits.mplot3d import Axes3D
from itertools import product, combinations


fig = plt.figure(100)
ax = fig.add_subplot(111, projection='3d')
# Parameterisierung der Kugeloberfläche mit Längen- und Breitengraden
phi = np.linspace(0,2*np.pi,20).reshape(-1,1)
theta = np.linspace(0,np.pi,20).reshape(1,-1)

x = np.cos(phi) * np.sin(theta)
y = np.sin(phi) * np.sin(theta)
z = np.ones_like(phi) * np.cos(theta)

ax.plot3D(x.flatten(),y.flatten(),z.flatten(),'r')
my_col = plt.cm.hsv((x+1)/2)
ax.plot_surface(x,y,z,facecolors=my_col)

# Würfel
r = [-1, 1]
for s, e in combinations(np.array(list(product(r,r,r))), 2):
    if np.sum(np.abs(s-e)) == r[1]-r[0]:
        ax.plot3D(*zip(s,e), color="b")

ax.set_aspect('equal')
```



Was passiert bei der Konstruktion des Würfels?

```
[xx for xx in product(r,r,r)]
```

```
[(-1, -1, -1),
 (-1, -1, 1),
 (-1, 1, -1),
 (-1, 1, 1),
 (1, -1, -1),
 (1, -1, 1),
 (1, 1, -1),
 (1, 1, 1)]
```

2er Kombinationen

```
[xx for xx in combinations( [1,2,3],2)]
```

```
[(1, 2), (1, 3), (2, 3)]
```

Zip-Befehl

```
[xx for xx in zip((-1, -1, -1), (1, 2, 3))]
```

```
[(-1, 1), (-1, 2), (-1, 3)]
```

Ist das gleiche wie oben

```
xx = [*zip((-1, -1, -1), (1, 2, 3))]
xx
```

```
[(-1, 1), (-1, 2), (-1, 3)]
```