```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# """
# Created on Fri Jan 12 09:34:34 2018
#
# @author: christianehelzel
# """
import numpy as np
import matplotlib.pyplot as plt
```

1 Eine Einführung in die objektorientierte Programmierung

Eine ausführliche Beschreibung finden Sie hier:

www.python-kurs.eu/python3_klassen.php

Wir haben bereits oft vordefinierte Objekte und Methoden von Klassen benutzt, ohne diese Konstrukte in unseren eigenen Programmen zu verwenden.

1.1 Beispiel für die Nutzung von Klassen:

Die numpy.array-Klasse

```
A = np.array([[1., 2.],[3.,4.]]) # A ist ein Array / A ist ein Objekt der array-Klasse
```

Attribute der numpy.array Klasse:

```
A.shape
A.dtype
```

```
dtype('float64')
```

Methoden der numpy.array Klasse:

```
A.min()
A.max()
A.transpose() # oder in Kurzform A.T
A.flatten()
```

```
array([ 1.,  2.,  3.,  4.])
```

Eine Klasse definiert einen neuen Datentyp.

Wir wollen nun selbst einen neuen Datentyp Student definieren.

Die minimale Definition einer Klasse in Python hat die Form

```
class Student:
    pass # pass gibt an, dass hier noch irgendwas hin soll
```

x und y sind Objekte des Datentyps Student

```
x = Student()
y = Student()
```

Die Studenten x, y haben bisher keinerlei Eigenschaften (Attribute). Wir wollen Ihnen einen Namen und eine Matrikelnummer geben

Dazu benutzen wir die __init__-Methode

```
class Student:

    def __init__(self,first,last,nr):
        self.first = first
        self.last = last
        self.nr = nr
```

__init__ wird automatisch beim Erstellen eines Objektes aufgerufen

```
x = Student('Mia','Schmidt',101)
y = Student('Max','Müller',102)
```

Wir können der Klasse Student auch eine email-Adresse zuordnen, die automatisch bein Erstellen eines Objektes generiert wird:

```
class Student:

    def __init__(self,first,last,nr):
        self.first = first
        self.last = last
        self.nr = nr
        self.email = first + '.' + last + '@hhu.de'


x = Student('Mia','Schmidt',101)
y = Student('Max','Müller',102)
```

Wir wollen nun Vor- und Nachnamen sowie die e-mail Adresse von Student x ausgeben. Zwischen dem Namen und der e-mail Adresse soll ein Komma stehen.

```
print('{} {}, {}'.format(x.first,x.last,x.email))
```

```
Mia Schmidt, Mia.Schmidt@hhu.de
```

Falls wir diese Informationen mehrfach ausgeben wollen, dann ist es effizienter, dazu eine Methode der Klasse Student zu definieren.

```
class Student:

    def __init__(self,first,last,nr):
```

```
        self.first = first
        self.last = last
        self.nr = nr
        self.email = first + '.' + last + '@hhu.de'

    def print_name_mail(self):
        return '{} {}, {}'.format(x.first,x.last,x.email)

x = Student('Mia','Schmidt',101)
y = Student('Max','Müller',102)


x.print_name_mail()
y.print_name_mail()
```

```
'Mia Schmidt, Mia.Schmidt@hhu.de'
```

Rules regarding self (from Lantangen's book, page 419):

1) Any class method must have self as first argument. (The name can be any valid variable name, but the name self is a widely established convention in Python.)
2) self represents an (arbitrary) instance of the class.
3) To access any class attribute inside class methods, we must prefix with self, as in self.name, where name is the name of the attribute.
4) self is dropped as argument in calls to class methods.

1.2 Ein mathematisches Beispiel

Wiederholung: In VL2 haben wir eine Funktion zur Berechnung der Bahnkurve eines Balls betrachtet

```
def h(x):
    import numpy as np

    g = 9.81     # Gravitationsbeschleunigung
    v0 = 20.     # Anfangsgeschwindigkeit
    theta = 45. # Winkel mit x-Achse (in Grad)
    y0 = 2.      # Anfangsposition des Balls ist (0,y0)

    theta = theta*np.pi/180

    return x*np.tan(theta) \
    - g*x**2 /(2*v0**2 * np.cos(theta)**2) + y0 # Höhe des Balls
```

Nun betrachten wir eine Lösung unter Verwendung von Klassen

```
class Y:
#     """
#     Horizontale Bewegung eines Balls
#
#     Attribute: v0 Anfangsgeschwindigkeit, theta Abwurfwinkel, y0 Höhe beim Abwurf
```

```
#
#     Methoden: value(x): berechnet Höhe als Funktion von x
#     """

    def __init__(self,v0,theta,y0):
        import numpy as np

        self.v0 = v0
        self.theta = theta*np.pi/180
        self.y0 = y0
        self.g = 9.81

    def value(self,x):
        import numpy as np

        y = x*np.tan(self.theta) \
    - self.g*x**2 /(2*self.v0**2 * np.cos(self.theta)**2) + self.y0

        return y
```

y1 und y2 sind zwei Objekte der Klasse Y

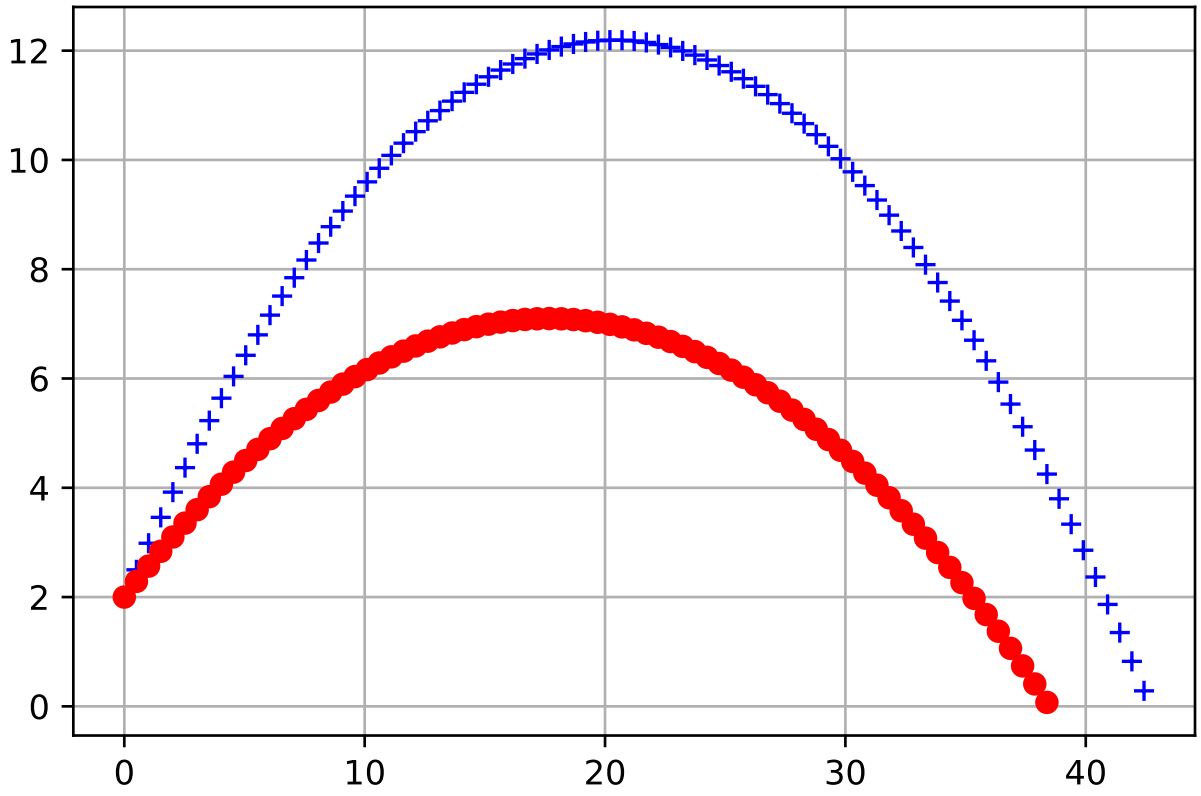
```
y1 = Y(20.,45.,2.)
y2 = Y(20.,30.,2.)

y1.value(10)
y2.value(10.)
```

```
6.1385026918962575
```

Wir können nun mit y1.value und y2.value wie mit typischen Python Funktionen arbeiten

Wir können die Funktionen plotten

```
x = np.linspace(0,50,100)
y1plt = y1.value(x)
y1plt[y1plt<0] = None
plt.plot(x,y1plt,'b+')

y2plt = y2.value(x)
y2plt[y2plt<0] = None
plt.plot(x,y2plt,'ro')
plt.grid('on')
```



Wir können auch den Differenzenquotienten unter Verwendung einer Funktion berechnen

```
def diff(f,x,h=1E-5):
    return (f(x+h)-f(x-h))/(2*h)

diff(y1.value,10.)
```

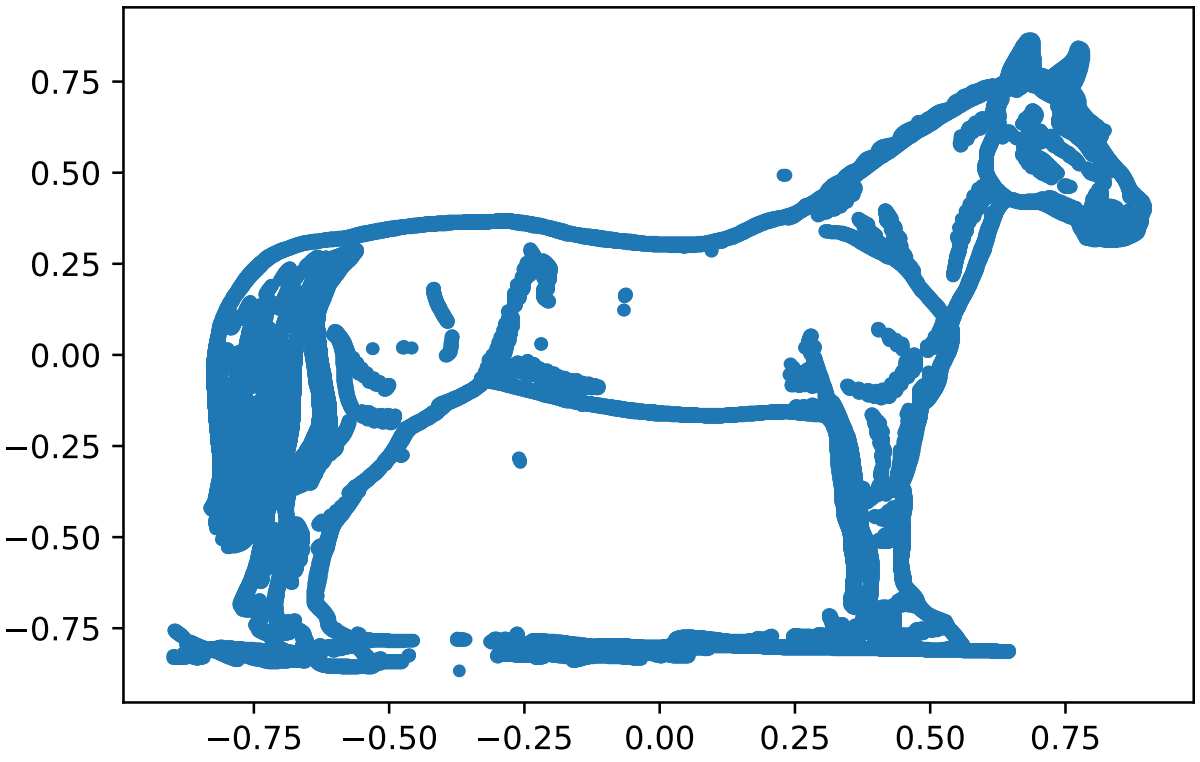
```
0.50950000005656193
```

Nun betrachten wir ein Beispiel aus der linearen Algebra

(Dieser kurze Abschnitt dient dabei nur der Vorbereitung)

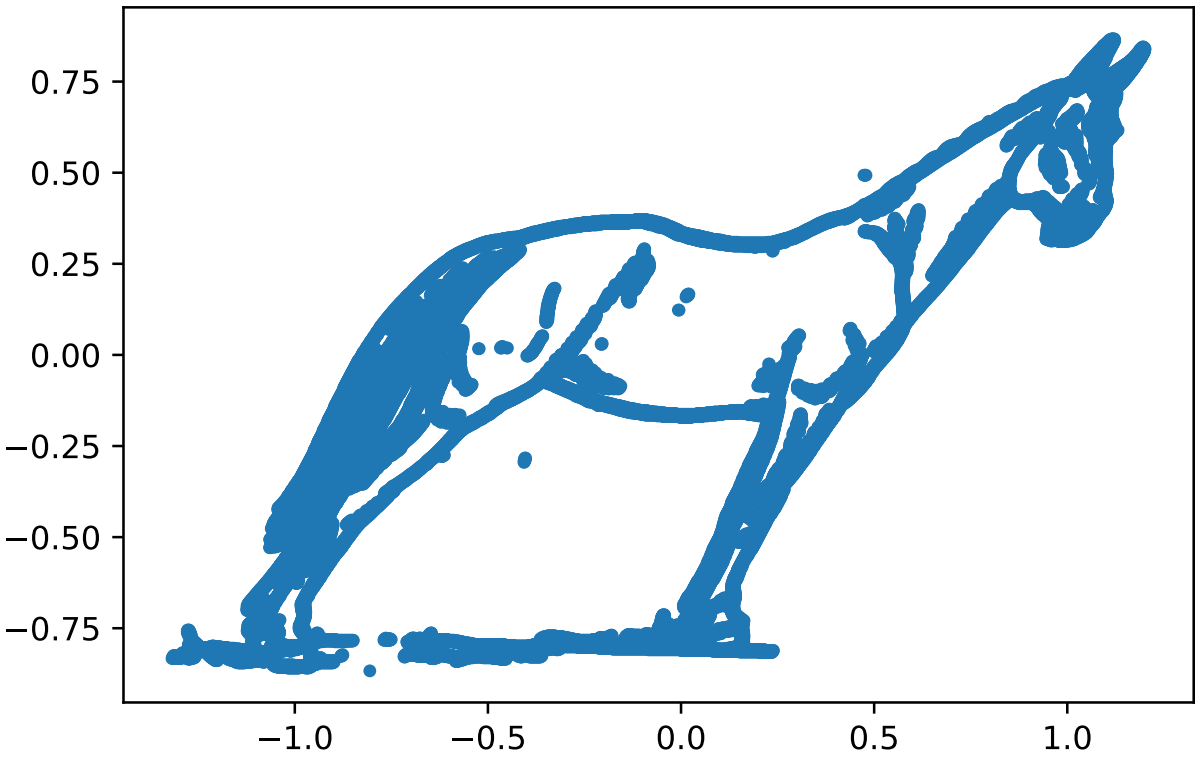
Lade numpy-Array

```
H = np.load("horse.npy")              # lade numpy-array
plt.figure()
plt.plot(H[0,:],H[1,:],'.')
plt.show()
```



Transformieren

```
A = np.array([[1.,0.5],[0.,1.]])     # definiere 2x2 Matrix A
H = A@H                              # Betrachte für jeden Punkt (x,y) aus
                                     # den Spalten von H die Abbildung
                                     # A @ (x,y)^T
plt.figure() # Zeichne die transformierte Punktmenge
plt.plot(H[0,:],H[1,:],'.')
plt.show()
```



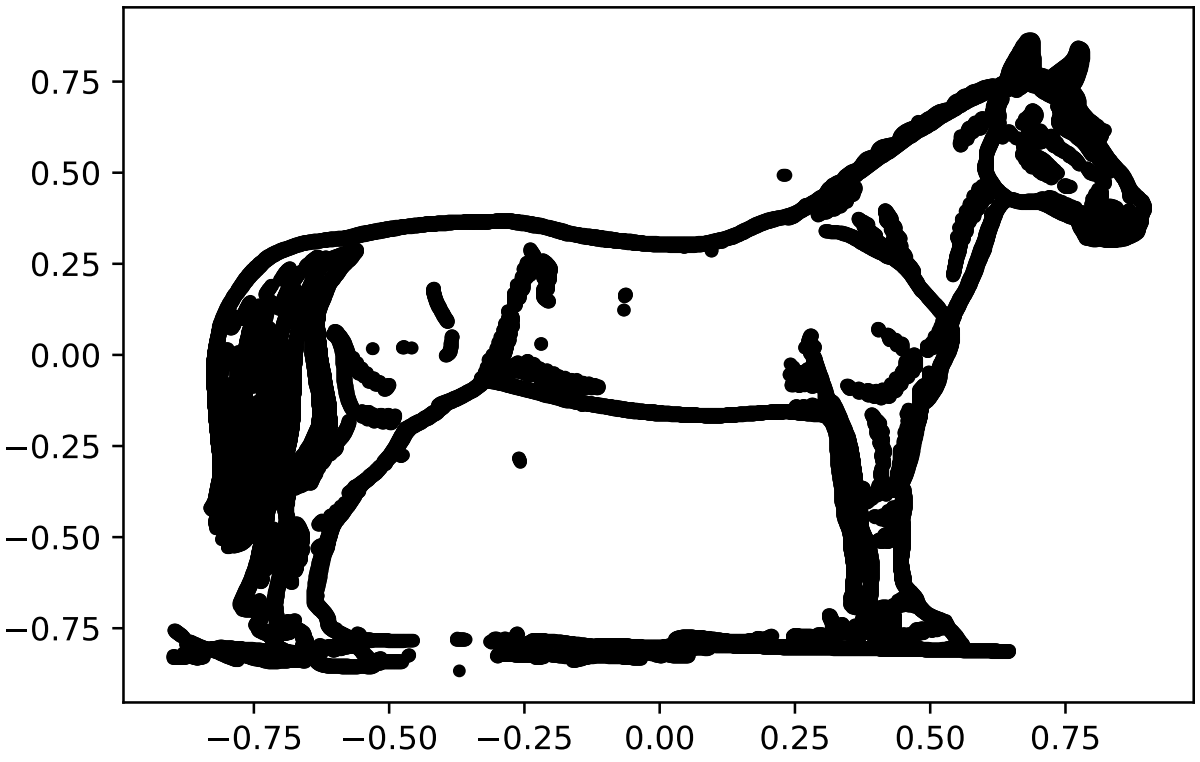
Wir betrachten nun eine Klasse zur Berechnung verschiedener Transpormationen

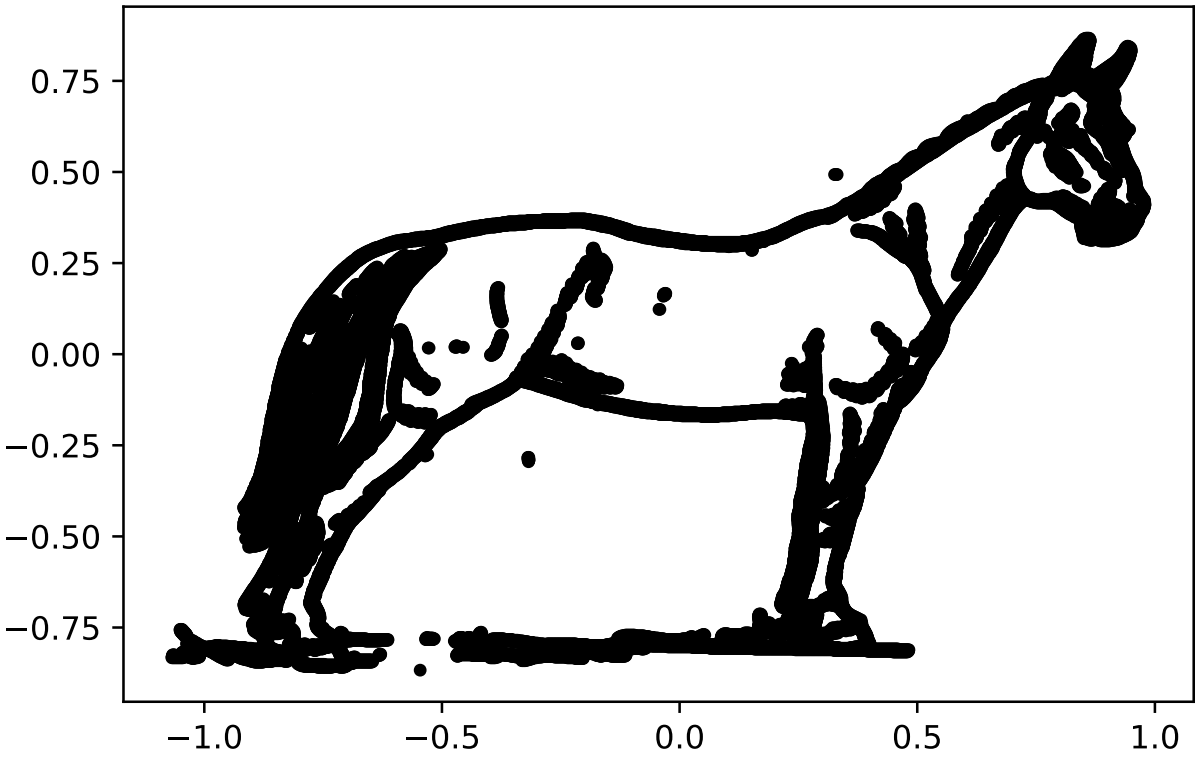
```
class ImageTransformation():
#     '''
#     Die Klasse ImageTransformation lädt aus einer Datei eine Punktmenge
#     und führt verschiedene Transformationen durch
#     '''
    import numpy as np

    def __init__(self,fname):

        self.A = np.load(fname)

    def stretch(self,a,b):

        M = np.array([[a,0.],[0.,b]])
        return M@self.A

    def shear(self,a):

        M = np.array([[1.,a],[0.,1.]])
        return M@self.A

    def reflection(self,a,b):

        M = np.array([[a**2-b**2, 2.*a*b],[2.*a*b, b**2-a**2]])/(a**2 + b**2)
        return M@self.A

    def rotation(self,theta):

        M = np.array([[np.cos(theta),-np.sin(theta)],[np.sin(theta), np.cos(theta)]])
        return M@self.A
```

```
    def display(self,M=None):
        import matplotlib.pyplot as plt

        if M is None:
            M = self.A

        fig = plt.plot(M[0,:],M[1,:],'k.')
        return fig
```

Zeichnen

```
plt.figure()
horse = ImageTransformation("horse.npy")
fig1=horse.display()
plt.figure()
#horse.display(horse.rotation(np.pi/3.))
fig2=horse.display(horse.shear(0.2))
```


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75


0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0