Minimaleinführung in Python 3

Python ist eine einfach zu lernende, aber mächtige Programmiersprache mit effizienten abstrakten Datenstrukturen und einem einfachen, aber effektiven Ansatz zur objektorientierten Programmierung. Durch die elegante Syntax und die dynamische Typisierung ist Python als interpretierte Sprache sowohl für Skripte als auch für schnelle Anwendungsentwicklung (Rapid Application Development) hervorragend geeignet.

Bibliotheken

Es gibt eine große Anzahl von verschiedenen Python Bibliotheken, bspw. numy, scipy, eigen, ... Je nachdem fuer welchen Zweck man gerad eine Bibliothek benötigt, muessen diese geladen werden. Dies erfolgt im Allgemeinen durch einen "import" Befehl

1. import BIBLIOTHEK as NAME

Will man nur auf eine Funktion der Bibliothek zugreifen, so verwenden wir

2. from BIBLIOTHEK import NAME

In [1]:

```
# Laden der Bibliothek
import numpy as np # lädt die Bibliothek, allerding muss jedesmal np miteingegeb
en werden

# Spezifische Funktion
from numpy.linalg import eig

from numpy import * # lädt alle funktion in der Bibliothek
```

Variablen und Operatoren

Variablen

In [2]:

```
# integer (ganze Zahl)
a = 1
# float (Fließkommazahl)
b = 3.14159
# string (zeichenkette)
c = "Hello world!"
```

In [3]:

```
# der Typ einer vaiablen kann we folgt abgefragt werden
print( type(a) )
print( type(b) )
print( type(c) )
```

```
<class 'int'>
<class 'float'>
<class 'str'>
```

In [4]:

```
# Operatoren: +,-,*,/
a1 = 1
a2 = 2
A = a1 + a2
print(type(A))


b1 = 1.1
b2 = 2.1
B = b1 + b2
print (type(B))

# Addition von int + float = float
print (type(A+B))

# Vorsicht bei int division
print(1/2)  # in python2  1/2 = 0
```

```
<class 'int'>
<class 'float'>
<class 'float'>
0.5
```

Sonstiges

In [5]:

```
# soll der user eine Zahl vorgeben, so verwenden wir den Befehl input()
c = input()
```

```
4
```

In [6]:

```
print( int(c)+1 )
```

```
5
```

In [7]:

```
# zwei strings können ebenfalls zusammengefasst werden
s1 = "Hello "
s2 = "World!"

s = s1+s2
print (s)
```

```
Hello World!
```

Zufallszahlen

Zufallszahlen können durch importieren der richtigen Bibliothek erhalten werden

In [8]:

```
import random as rd
```

In [9]:

```
# Zufalls-int zwischen 1 und 10
rd.randint(1,10)
```

Out[9]:

```
10
```

In [10]:

```
# Zufalls -loat  in  [0.0, 1.0).
rd_fl = rd.random()
rd_fl
```

Out[10]:

```
0.1118963831373645
```

Listen

Listen werden durch [ ] erstellt. Sie können mit allen Variablen Typen befüllt werden!

In [11]:

```
# erstelle eine leere Liste
list1 = []
```

In [12]:

```
# listen werden wie folgt gefüllt
list1.append(3)              # int
list1.append(3.14)           # float
list1.append("Hello world!") # string
```

In [13]:

```
list1
```

Out[13]:

```
[3, 3.14, 'Hello world!']
```

In [14]:

```
# Listenelemente werden wie folgt aufgerufen
print (list1[0])
print (list1[1])
print (list1[2])
```

```
3
3.14
Hello world!
```

In [15]:

```
# listen werden wie folgt angehängt
list2 = [1.1,0.1,0.3]
list3 = [2.1,1.1,1.3]
list4 = list2 + list3
```

In [16]:

```
list4
```

Out[16]:

```
[1.1, 0.1, 0.3, 2.1, 1.1, 1.3]
```

Zufallszahlen 2

In [17]:

```
import random as rd
```

In [18]:

```
# Liste mit Zufallszahlen füllen
rd_list = [rd.random() for i in range(4)]
rd_list
```

Out[18]:

```
[0.5343869891706686, 0.3107816622058477, 0.757548747296998, 0.560391
7591722383]
```

In [19]:

```
# beliebiges Element ausgeben
rd.choice(rd_list)
```

Out[19]:

```
0.757548747296998
```

In [20]:

```
# Liste mischen
rd.shuffle(rd_list)
rd_list
```

Out[20]:

```
[0.5343869891706686, 0.5603917591722383, 0.3107816622058477, 0.75754
8747296998]
```

If, else, elif

In [21]:

```
# variable
num = 3
```

In [22]:

```
if num==1 :
    print ("wert = ", num)
elif num==2:
    print( "wert = ", num)
else:
    print( "weder noch")

#Befehle: > , < , !=
```

```
weder noch
```

For, while

In [23]:

```
# for schleife
for i in range(3):
    print( i )
```

```
0
1
2
```

In [24]:

```
# while schleife
k=0
while(k<3):
    print( k)
    k += 1
```

```
0
1
2
```

Dictionaries

Ein Dictionary ist ein assoziatives Feld. Ein Dictionary besteht aus Schlüssel-Objekt-Paaren. Zu einem bestimmten Schlüssel gehört immer ein Objekt. Man kann also die Schlüssel auf die Objekte "abbilden", daher der Kategorienname Mapping.

Wir wollen uns die an einen einfachen Beispiel dies veranschaulichen:

In [25]:

```
#  Erstellen und Befüllen eines Dictionaries
dict1 = {"house" : "Haus", "cat":"Katze", "black":"schwarz"}
```

In [26]:

```
# Ausgabe
print (dict1["house"])
print (dict1["cat"])
print (dict1["black"])
```

```
Haus
Katze
schwarz
```

In [27]:

```
# soll nun ein Schlüssel samt Element gelöscht werden, gilt
del dict1["house"]
```

In [28]:

```
dict1
```

Out[28]:

```
{'black': 'schwarz', 'cat': 'Katze'}
```

In [29]:

```
# alternative kann ein Dictionarie auch wie folgt verwendet werden
dict_results = {}
dict_results["x_val"] = [1.,2.,3.,4.]   # x werte
dict_results["y_val"] = [1.,2.,3.,4.]   # f(x) = x werte
dict_results["dy_val"] = [1.,1.,1.,1.]  # ableitung f'(x) = 1
```

In [30]:

```
dict_results
```

Out[30]:

```
{'dy_val': [1.0, 1.0, 1.0, 1.0],
 'x_val': [1.0, 2.0, 3.0, 4.0],
 'y_val': [1.0, 2.0, 3.0, 4.0]}
```

Numpy und Scipy

Werden verschiedenen Funktionen, wie etwa $\sin$, $\cos$, etc. benoetigt, so bietet es sich an, die "scipy", bzw. "numpy" bibliothek zu verwenden. Neben verschiedenen mathematischen Funktionen bietet diese:

1. mathematische Funktion

2. arrays, vektoren, matrizen

3. Lineare Algebra Algorithmen (Determinante, Eigenwerte, ...)

4. ...

In [31]:

```
# importieren der Bibliothek
import numpy as np
import scipy as scp # gleiche syntax wie numpy (Vorsicht: siehe website)
```

Einfache Funktionen

In [32]:

```
np.sin(np.pi)
```

Out[32]:

```
1.2246467991473532e-16
```

In [33]:

```
np.cos(np.pi)
```

Out[33]:

```
-1.0
```

In [34]:

```
np.exp(0)
```

Out[34]:

```
1.0
```

Vektoren und Matrizen

In [35]:

```
# Vektoren
vec_1 = np.array([1,1,1])
vec_2 = np.array([2,2,2])
```

In [36]:

```
np.dot(vec_1,vec_2)
```

Out[36]:

```
6
```

In [37]:

```
vec_1+vec_2
```

Out[37]:

```
array([3, 3, 3])
```

In [38]:

```
# Matrizen
mat_1 = np.array([[1,0,3],
                  [0,2,0],
                  [3,0,4]])

mat_2 = np.array([[4,0,5],
                  [0,2,0],
                  [5,0,5]])
```

In [39]:

```
np.dot(mat_1,mat_2)
```

Out[39]:

```
array([[19,  0, 20],
       [ 0,  4,  0],
       [32,  0, 35]])
```

In [40]:

```
mat_1+mat_2
```

Out[40]:

```
array([[5, 0, 8],
       [0, 4, 0],
       [8, 0, 9]])
```

In [41]:

```
# Erstelle eine leere Matrix oder einen leeren Vektor
mat_zero = np.zeros([3,3])
vec_zero = np.zeros([3])
```

In [42]:

```
mat_zero
```

Out[42]:

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

In [43]:

```
vec_zero
```

Out[43]:

```
array([0., 0., 0.])
```

In [44]:

```
# Auf Matrizen und Vektoren wird wie folgt zugegriffen
print (mat_1[2,2])
print (vec_1[0])
```

```
4
1
```

In [45]:

```
# Leere Matrizen oder Vektoren könne dann wie fogt befüllt werden
vec_zero[0] = 1.1
mat_zero[0,0] = 1.1
```

In [46]:

```
vec_zero
```

Out[46]:

```
array([1.1, 0. , 0. ])
```

In [47]:

```
mat_zero
```

Out[47]:

```
array([[1.1, 0. , 0. ],
       [0. , 0. , 0. ],
       [0. , 0. , 0. ]])
```

Lineare Algebra

In [48]:

```
# Eigenwerte und -vektoren
eigval, eigvec = np.linalg.eig(mat_1)
```

In [49]:

```
eigval
```

Out[49]:

```
array([-0.85410197,  5.85410197,  2.        ])
```

In [50]:

```
eigvec
```

Out[50]:

```
array([[-0.85065081, -0.52573111,  0.        ],
       [ 0.        ,  0.        ,  1.        ],
       [ 0.52573111, -0.85065081,  0.        ]])
```

In [51]:

```
# test: ob das auch wirklich stimmt D = S^-1 * mat_1 * S
np.dot( np.linalg.inv(eigvec) , np.dot(mat_1, eigvec) )
```

Out[51]:

```
array([[-8.54101966e-01,  8.88178420e-16,  0.00000000e+00],
       [ 4.99600361e-16,  5.85410197e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  2.00000000e+00]])
```

Funktionen

Wir wollen mittels der Boxregel (siehe Aufgabenblatt 4) numerisch ein Integral bestimmen, naemlich \begin{align} \int_{1}^{4} f(x) \, \mathrm d x \quad \text{und} \quad f(x) = x^3+2x^2-2. \end{align} Wir wissen \begin{align} \int_{1}^{4} f(x) \, \mathrm d x = 99.75. \end{align}

In [52]:

```
# Alle Funktionen sind 1D

# Definition einer simplen Funktion
def fun(x):
    return x**3+2*x**2-2

# Definition der Boxregel (Funktion mit Funktion als Input)
def boxrule(a,b,function):
    return (b-a)*function(b)

# Berechnung der gesamten Fläche einer Funktion mittel Boxregel
def integrate_intervall(a,b,h,fun):
    steps = int((b-a)/h)
    value = 0.0
    for i in range(steps):
        value += boxrule(a+i*h,a+(i+1)*h,fun)

    return value
```

Um herauszufinden, wie gut die Integration von h (Integrationsschritt) abhängt, betrachten wir eine Liste von unterschiedlichen h. Diese wollen wir später plotten.

In [53]:

```
# Erstellen einer List und der Differenzfunktion

# anlytischer Ausdruck: 99.75
val_an = 99.75

h = [0.1,0.01,0.001,0.0001, 0.00001]
dif = []
for i in h:
    dif.append(integrate_intervall(1,4,i,fun)-val_an)
```

In [54]:

```
dif
```

Out[54]:

```
[4.697499999999991,
 0.4654750000000263,
 0.046504750000167405,
 0.004650047499723087,
 0.0004650004746906688 5]
```

Plotten

Zum Plotten verwenden wir eine neue Bibliothek von Python, nämlich "matplotlib". Diese wird zunächst importiert, sowie der Plottbefehl. Wir wollen sowohl $f(x)$, als auch die Differenz plotten.

In [55]:

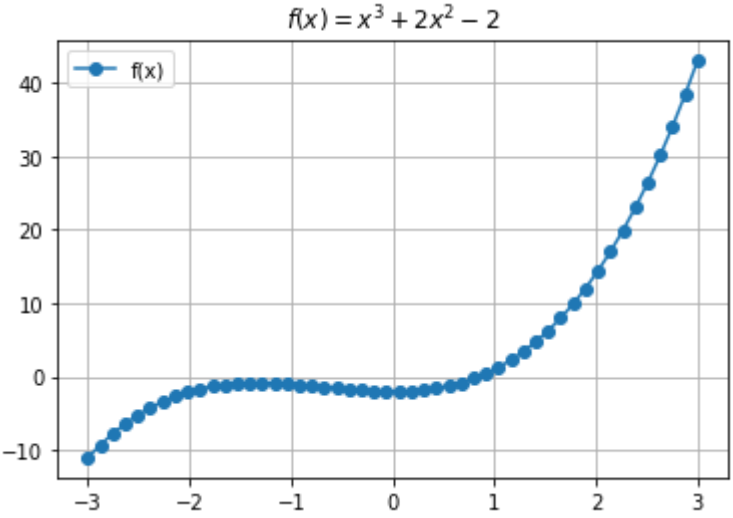
```
# Iportieren der richtigen bibliothek
#import matplotlib
import matplotlib.pyplot as plt
# wird nur in jupyter benoetigt
%matplotlib inline
```

In [56]:

```
# Erstellen von 2 Listen fuer den Funktionsplot
x = np.linspace(-3,3,50) # oder: [-3+6.0*i/50 for i in range(50)]
y = fun(x) # oder: [fun(i) for i in x]
```

In [57]:

```
# Einfacher Plot
plt.plot(x,y, "-o", label='f(x)')
plt.title('$f(x) = x^3+2x^2-2$')
plt.legend()
plt.grid()
plt.show()
```



In [58]:

```
# Plotten der Differenzen (Semilogarithmisch)
plt.semilogx(h,dif, "-o", label='difference')
plt.title('Semilogplot')
plt.legend()
plt.grid()
plt.show()
```