Tutorium

November 1, 2021

[8]:
```
import numpy as np
import time
import matplotlib.pyplot as plt
```

Basic Basics

if-Bedingungen

[ ]:
```
#if statement:
#    wenn statement zutrifft --> mache was!
a = 1
b = 2
if b > a: #In der If-Bedingung muss immer True oder False rauskommen!
    print("b ist größer als a")


#Weitere Verzweigungen
a = 1
b = 2
if b > a:
    print("b ist größer als a")

elif b < a:
    print("b ist größer als a")

#Else Bedingung (wenn nichts zutrifft)
a = 1
b = 2
if b > a:
    print("b ist größer als a")

elif b < a:
    print("b ist größer als a")

else:
    print("b und a sind gleich")
```

for loops

[12]:
```
#for loops brauchst du immer wenn du über sachen iterieren willst (also über␣
↪alle elemente einer list zb)
loop_liste = [1, 2, 3, 8, 9, 10]

for zahl in loop_liste:
    print(zahl)

#was nach dem "in" steht muss aber nicht unbedingt eine liste sein
s = "hallo"

for zeichen in s:
    print(zeichen)

#Sehr oft braucht man dann die range() funktion
#range(start, ende, steps)

for zahl in range(0, 10): #Achte darauf, dass 10 nicht inkludiert wird
    print(zahl)
```

while loop

[ ]:
```
#while statement:
#    mach was
#Solange also das statement True ist werden die Befehle in der while loop␣
↪ausgeführt

#dieses while loop macht genau das gleiche wie die letzte for loop von oben
n = 0
while n < 10:
    print(n)
    n = n + 1
```

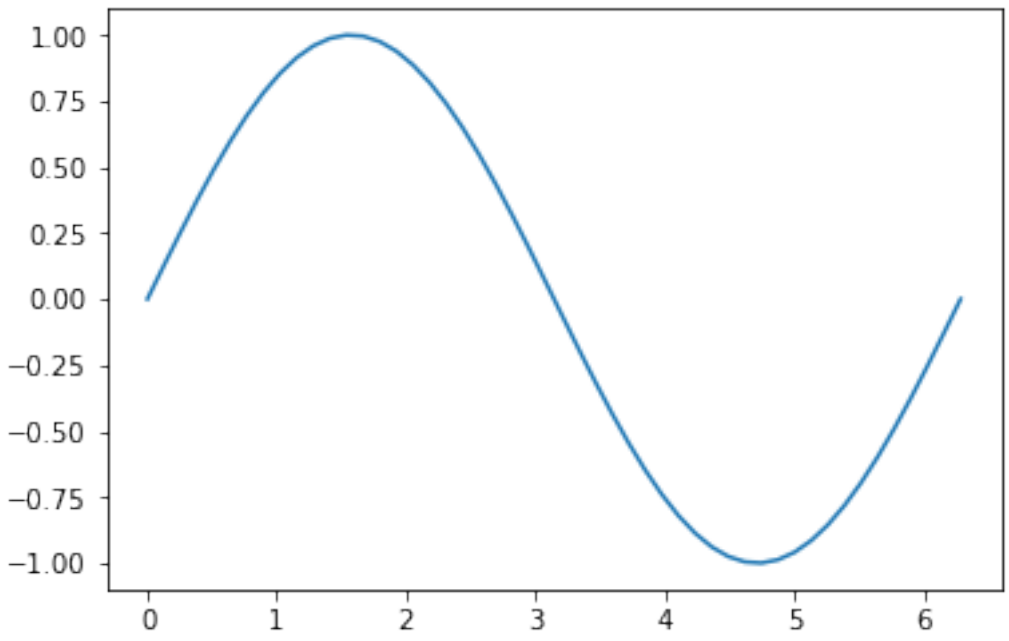
Funktionen

[ ]:
```
#Oftmals musst die Code-Segmente sehr oft ausführen (zb. summieren, ist das␣
↪eine Primzahl?, etc)
#Diese Code-Segmente kann man zusammenfassen in einer Funktion

def name_der_funktion(parameter1, parameter2): #Du kannst die parameter␣
↪natürlich benennen wie du willst
    print(parameter1, parameter2) #Wichtig parameter1 und parameter2 sind dann␣
↪nur in der Funktion definiert!


#Hier als bsp eine einfache funktion die die summe von einer zahl n bis m bildet
def summe(n, m):
    s = 0
```

```
    for zahl in range(n, m+1):
        s = s + zahl

    return s

s = summe(1, 4)
```

Mach dir die Dinge einfach!

[13]:
```
#Es gibt tausende Funktionen die oft genau das machen was du eigentlich brauchst
#Zb die summe() funktion von oben bracuhst du gar nicht

s = sum([1, 2, 3, 4]) #Spart zeit und der code schaut gleich viel␣
 ↪übersichtlicher aus (und er ist auch schneller)

#Andere nützliche funktionen sind zb:
ma = max([1, 2, 3, 4]) #Gibt den maximalen Wert aus der liste zurück
mi = min([1, 2, 3, 4]) #ich glaub du hast es verstanden

#Numpy ist hier sehr mächtig und hat extrem viel was du fürs studium brauchst
pi = np.pi

x = np.linspace(0, 2*pi)
plt.plot(x, np.sin(x)) #Oder plot funktionen aus matplotlib
plt.show()
```



Auch nice: break, continue

[ ]:
```
#Oftmals will man eine Schleife unter einer bestimmten bedingung abrechen

x = 0
while True: #Endlosschleife
    x = x + 1
    if x > 10:
        break #Die schleife wird abgebrochen

#ich komme also nach dem break wieder hier hin und gehe weiteren code durch

#oft will man nicht unter bestimmten bedingen den ganzen code in einer schleife␣
 ↪durchführen

for i in range(0, 10):
    if i%2 == 0:
        continue #Wenn ich hier hinkomme gehe ich die schleife weiter ohne den␣
 ↪code darunter auszuführen

    print(i)
```

Basics

Aufgabe 1

[10]:
```
i = 1
d = 1.1

print(type(i), type(d))

a = 3
b = 4

print(a//b)
print(a/b)

s = '15'
x = int(s)
print(x, type(x))
print(s, type(s))
```

```
(<type 'int'>, <type 'float'>)
0
0
(15, <type 'int'>)
('15', <type 'str'>)
```

Aufgabe 2

[3]:
```
point_1 = [2.8, -4.7, 0.4]
point_2 = [-8.1, 3.0, -10.6]

s = 0
for i in range(len(point_1)):
    s += (point_1[i] - point_2[i])**2

print(np.sqrt(s))
```

```
17.294507798720378
```

Mittel

Aufgabe 6

[4]:
```
def dot(v1, v2):

    if len(v1) != len(v2):
        return None

    s = 0
    for i in range(len(v1)):
        s = s + v1[i] * v2[i]

    return s


def cross(v1, v2):

    return [v1[1]*v2[2] - v1[2]*v2[1],
            v1[2]*v2[0] - v1[0]*v2[2],
            v1[0]*v2[1] - v1[1]*v2[0]]


point_1 = [2.8, -4.7, 0.4]
point_2 = [-8.1, 3.0, -10.6]

print(dot(point_1, point_2))
print(cross(point_1, point_2))
```

```
-41.02
[48.62, 26.439999999999998, -29.67]
```

Aufgabe 7

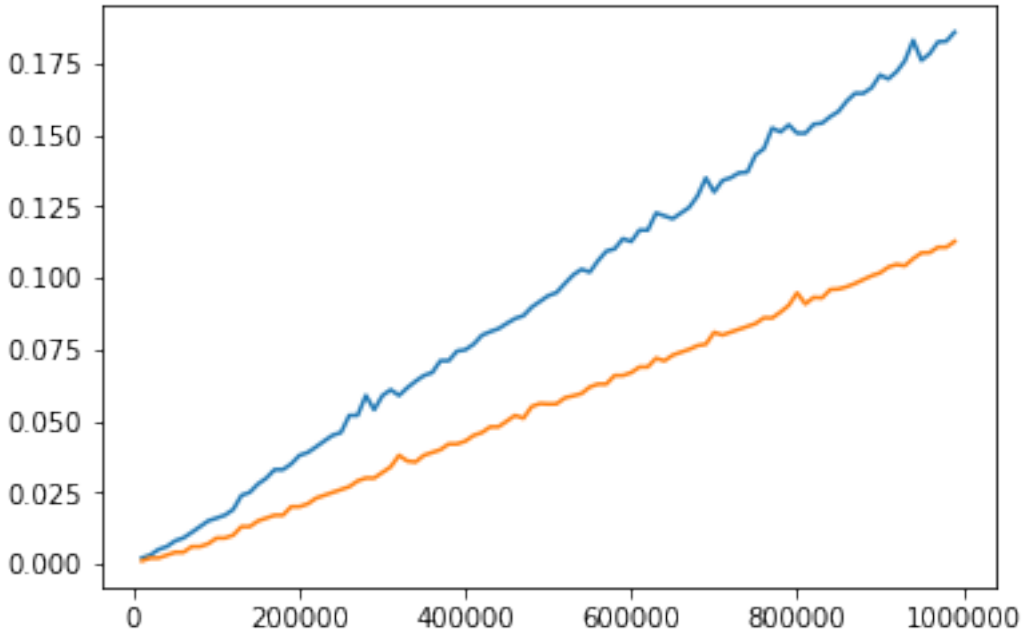
[7]:
```
N = 10**6

s_time = time.time()
```

```
daten  = []

for i in range(0, N):
    daten.append(i**0.5)

loop_time = time.time() - s_time

s_time  = time.time()
daten   = np.sqrt(np.arange(0 ,N))
np_time = time.time() - s_time


print('Time for loop:', loop_time)
print('Time for np:', np_time)
```

```
('Time for loop:', 0.22129607200622559)
('Time for np:', 0.010730981826782227)
```

Aufgabe 8

[58]:
```
def sqrt_loop_time(N):
    s_time = time.time()
    [i**0.5 for i in range(0, N)]

    return time.time() - s_time

def sqrt_np_time(N):
    s_time = time.time()
    np.sqrt(range(0 ,N))

    return time.time() - s_time


def plot_time(N_max, method):
    Ns = np.arange(10000, N_max, 10000)
    times = np.zeros(len(Ns))

    for i in range(len(times)):
        times[i] = method(Ns[i])

    plt.plot(Ns, times)

plot_time(10**6, sqrt_loop_time)
plot_time(10**6, sqrt_np_time)
```



Aufgabe 10

[66]:
```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(a[4:8])
print(a[::2])
print(a[::-1])
print(a[-1])
```

```
[5, 6, 7, 8]
[1, 3, 5, 7, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
10
```

Advanced

Aufgabe 1

[204]:
```
def remove_duplicates(array, thres=10):
    r_array = []
    dup_c   = 0

    for element in array:
        if element not in r_array:
            r_array.append(element)

        else:
            dup_c += 1
```

```
    if len(array) / len(r_array) * 100 < thres:
        raise ValueError('Threshold not met')

    return r_array, dup_c

test = np.random.randint(100, size=100)

f_list, dup_c = remove_duplicates(test)
print('Final list:', f_list)
print('Count of removed values:', dup_c)
```

```
Final list: [89, 44, 23, 88, 11, 94, 62, 91, 2, 10, 28, 5, 18, 49, 86, 97, 12,
59, 51, 98, 7, 53, 55, 8, 90, 77, 6, 58, 42, 75, 84, 27, 50, 39, 0, 80, 41, 4,
69, 83, 99, 93, 79, 87, 22, 30, 64, 3, 15, 81, 40, 78, 52, 46, 29, 14, 36, 21,
68, 43]
Count of removed values: 40
```

Aufgabe 2

[202]:
```
def isPrime(n):
    if n == 2 or n == 3:
        return True

    if n%2==0 or n < 2:
        return False

    #Nur über ungerade Zahlen iterieren
    for i in np.arange(3, int(np.sqrt(n))+1, 2):
        if n%i==0:
            return False

    return True

def generate_prime_list(N):
    prime_list = []

    for n in range(0, N):
        if isPrime(n):
            prime_list.append(n)

    return np.array(prime_list)

def prime_sum(thres):
    s = []                                          #List that contains␣
 ↪all the sums
    s_prime_list = []                               #List that contains␣
 ↪list of all primes added
```

```
    prime_list    = generate_prime_list(thres)

    for i in range(len(prime_list)):                          #Go through every␣
↪possible sum
        j = 2                                                 #Variable we need in␣
↪order to slice the prime list
        while i+j < len(prime_list):
            prime_sublist = prime_list[i:i+j]                 #Here we create the␣
↪sublists
            sum_prime     = np.sum(prime_sublist)

            if isPrime(sum_prime) and sum_prime < thres:      #Check if the sum is␣
↪a prime
                s.append(sum_prime)
                s_prime_list.append(prime_sublist)

            j += 1                                            #Reset j

    len_list = [len(l) for l in s_prime_list]
    i = np.where(len_list == np.max(len_list))[0][0]          #Find position where␣
↪len is max

    return s[i], s_prime_list[i], len_list[i]                 #Return the max sum,␣
↪the primes added and the len list

s, primes, l = prime_sum(1000)
print('Sum:', s, 'of', l, 'primes')
print('Added primes:', primes)
```

```
Sum: 953 of 21 primes
Added primes: [ 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89]
```

[ ]: